Bridging the Gap between a Behavioural Formal Description Technique and User Interface description language: Enhancing ICO with a Graphical User Interface markup language

Eric Barboni, Célia Martinie, David Navarre, Philippe Palanque, Marco Winckler

Bridging the gap between a behavioural formal description technique and a user interface description language: Enhancing ICO with a graphical user interface markup language

Eric Barboni, Célia Martinie, David Navarre *, Philippe Palanque, Marco Winckler

Institute of Research in Informatics of Toulouse, University of Toulouse, Interactive Critical Systems (ICS) Team, 118 route de Narbonne, 31042 Toulouse Cedex 9, France

HIGHLIGHTS

- We present the current state of the Interactive Cooperative Object (ICO) formal description technique and gaps to be bridged.
- We show how we succeed in adding means to ICO to fully describe a graphical user interface.
- We propose ways to integrate ICO within UsiXML.
- We use a case study of an interactive aircraft cockpit application.

ABSTRACT

In the last years, User Interface Description Languages (UIDLs) appeared as a suitable solution for developing interactive systems. In order to implement reliable and efficient applications, we propose to employ a formal description technique called ICO (Interactive Cooperative Object) that has been developed to cope with complex behaviours of interactive systems including event-based and multimodal interactions. So far, ICO is able to describe most of the parts of an interactive system, from functional core concerns to fine grain interaction techniques, but, even if it addresses parts of the rendering, it still not has means to describe the effective rendering of such interactive system. This paper presents a solution to overcome this gap using markup languages. A first technique is based on the Java technology called JavaFX and a second technique is based on the emergent UsiXML language for describing user interface components for multi-target platforms. The proposed approach offers a bridge between markup language based descriptions of the user interface components and a robust technique for describing behaviour using ICO modelling. Furthermore, this paper highlights how it is possible to take advantage from both behavioural and markup language description techniques to propose a new model-based approach for prototyping interactive systems. The proposed approach is fully illustrated by a case study using an interactive application embedded into interactive aircraft cockpits.

* Corresponding author. Tel.: +33 561556965.

E-mail addresses: barboni@irit.fr (E. Barboni), martinie@irit.fr (C. Martinie), navarre@irit.fr, navarre@univ-tlse1.fr (D. Navarre), palanque@irit.fr (P. Palanque), winckler@irit.fr (M. Winckler).

1. Introduction

In the recent years User Interface Description Languages (UIDLs) appeared to be a suitable solution for developing interactive systems as reported in many contributions such as [16,27] or [52]. These contributions highlighted the fact that, in order to provide a complete support for the development of interactive systems, UIDLs must address the three following different aspects of the User Interface (UI):

1. Description of the static structure of the user interfaces (i.e. presentation part which ultimately includes the description of user interface elements, e.g. widgets, and their composition);
2. Description of the dynamic behaviour (i.e. the dialogue part, describing the dynamic relationships between components including states in which the interface can be in, events to which the user interface is able to react to, actions that can be performed when events occur, and behavioural constraints such as preconditions or post conditions);
3. Definition of the presentation attributes (i.e. look & feel properties for rendering the UI elements).

Among the artefacts (specifications, models, prototypes …) produced during User Interface (UI) development, dynamic behavioural representations of the UI are one of the most misunderstood and one of the most difficult to exploit as detailed in [14,64]. Such behavioural representations can be used at various levels of the architecture of interactive systems (such as input device level, device driver level, interaction technique level or even dialogue level). Dialogue models play a major role during UI design by capturing the dynamic aspects of user interaction with the system which includes the specification of: relationship between presentation units (e.g. transitions between windows) as well as between UI elements (e.g. activate/deactivate buttons), events chain (i.e. including fusion/fission of events when multimodal interaction is involved) and integration with the functional core which requires mapping of events to actions according to predefined constraints enabling/disabling actions at runtime.

Problems and techniques related to the description of behavioural aspects of interactive systems have been extensively discussed in [47]. One of the outcomes of this analysis demonstrates that the ICO (Interactive Cooperative Object) formalism is one of the most expressive notations to describe the dynamic behavioural aspects of UIs. Indeed, ICO formalism is a formal description technique designed for the specification, modelling and implementation of interactive systems. ICO has been demonstrated efficient for describing several techniques including 3D and virtual reality [49], multimodal interaction techniques [38] and dynamic reconfiguration of interactive systems [48], while ICO models are executable and fully supported by the CASE tool PetShop [7] which has been shown effective for prototyping interactive techniques [46].

The software architecture of the Arch [9] model (illustrated in Fig. 1) makes it easier to understand how ICO is able to describe a user interface as a UIDL at various levels.

Fig. 1 highlights the basic components that must be addressed when dealing with the description of an interactive system. The dialogue component at the top of the arch is loosely coupled with implementation constraints, making it possible to describe it in a more abstract manner, while the physical interaction and the functional core are platform dependent. Physical interaction deals with a two way communication between the system and the user. On one side there is the communication from the system to the user, where the system state is rendered to the user through output devices and on the other side, the physical interaction allows a communication from the user to the system using input devices.

Several contributions have been built on top of ICO in order to support:

- Formal description of complex interactive systems behaviour [47].
- Formal description of inputs, supporting multimodal interaction description [38].
- Formal description of connexion to the functional core adapter and of the functional core itself [10].
- Formal description of the behaviour of interactive components, widgets, mechanisms such as picking [20] …and their link with the dialogue [8] making it possible to describe part of the rendering.

But even if rendering mechanisms are supported by ICO, as we already illustrated a first attempt to integrate SVG based rendering [57] mechanisms in previous work [6], ICO does not support the description of effective rendering (how data are rendered to the users…).

To overcome the lack of graphical user interface description, two solutions have been identified. A first solution using FXML (the JavaFX ™markup language [34] that supports the description of the graphical part of the user interface) has been studied due to its connection with Java technology on which PetShop CASE tool is also based. But in a longer term perspective, we envisioned that the synergistic use of UsiXML [40] and ICO should help in scaling up our approach to address problems such as cross platform deployment or adaptation in interactive systems. Indeed, UsiXML appears as an emergent standard for describing interactive systems, in particular the ones which are designed and developed to be deployed on multiple platforms [61].

In this paper we propose a first model-based approach to integrate a graphical user interface description using FXML and a behavioural description using ICO models. We then take advantage of this integration to look forward to extend the approach to UsiXML. By using ICO models it is possible to run the PetShop environment to control the execution of the application, making it possible to support prototyping not only of the user interface but also of the entire interactive system. Such approach has already been demonstrated efficient to model the behaviour of user interface components of cockpits of large civil aircrafts based on the standard ARINC 661 defined in [4,5]. Next section presents the current state of the ICO formal description technique and then presents the two main directions making ICO able to fully describe an

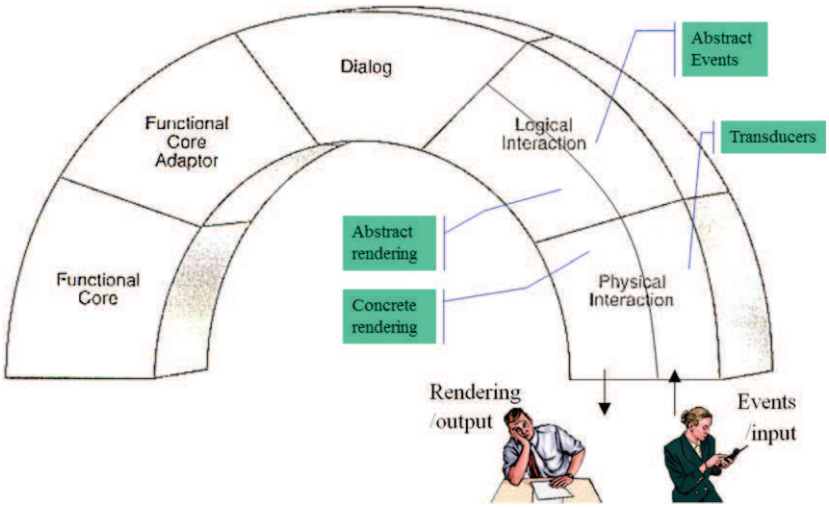


Fig. 1. The Arch model adapted from [9].

interactive system by systematic integration of the presentation part. More precisely, Section 2 presents an overview of ICO features and Section 3 presents works related to UIDL as well as a model-based design process. Section 3.3 illustrates the ICO formal description technique with its application to interactive cockpit applications, associates them to the corresponding block of the Arch architecture, and precisely shows up the need for a notation to describe the graphical rendering part of the UI. Section 5 then presents the integration of ICO and JavaFX while Section 6 presents how it would be possible to perform the same integration between ICO and UsiXML. Lastly, Section 7 concludes this paper, providing keys for future work.

2. ICO: an inventory of key features

In this section we provide an overview on the ICO capabilities for the modelling of user interfaces. We then discuss rendering aspects to highlight ICO weaknesses in providing effective rendering for the user interface. All features discussed here are illustrated within the case study (see Section 3.3).

2.1. The ICO User Interface Description Language: an overview

The Interactive Cooperative Object (ICO) formalism is based on concepts borrowed from the object-oriented approach (i.e. dynamic instantiation, classification, encapsulation, inheritance, and client/server relationships) to describe the structural or static aspects of systems, and uses high-level Petri nets [26] to describe their dynamics or behavioural aspects. In the ICO formalism, an object is an entity featuring five components: a cooperative object (CO), an available function, a presentation part and two functions (the activation function and the rendering function) that correspond to the link between the cooperative object and the presentation part.

The **Cooperative Object** (CO) models the behaviour of an ICO. It states (by means of a high-level Petri net) how the object reacts to external stimuli according to its inner state. Fig. 2 shows the concepts of the Cooperative Object models including: *places* (i.e. used as variables for tracking the system state), *transitions* (i.e. elements processing changes in the system state) and *arcs* (i.e. connecting places and transitions in a graph). *Arcs* can indicate input/output for tokens circulating in the graph; notice that an input arc (i.e. *InputArc*) can be extended to feature preconditions such as testing the availability of tokens in a place (*i.e. TestArc*) or preventing the movement of token accordingly to special conditions (i.e. *InhibitorArc*). The variables associated to an arc are expressed by the concept *EString*. Tokens can hold values of any class in the system. The types of tokens that can circulate in a given place are denoted through the relationship with the concept *EClass*.

The **presentation part** describes the external appearance of the ICOs. It is a set of widgets embedded into a set of windows. Each widget can be used for interacting with the interactive system (user interaction → system) and/or as a way to display information about the internal state of the object (system → user interaction).

The **activation function** (user inputs: user interaction → system) links users' actions on the presentation part (for instance, a click using a mouse on a button) to event services.

The **rendering function** (system outputs: system → user interaction) maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes through functions calls.

Additionally, an **availability function** is provided to link a service to its corresponding transitions in the ICO, i.e., a service offered by an object will only be available if one of its related transitions in the Petri net is available.

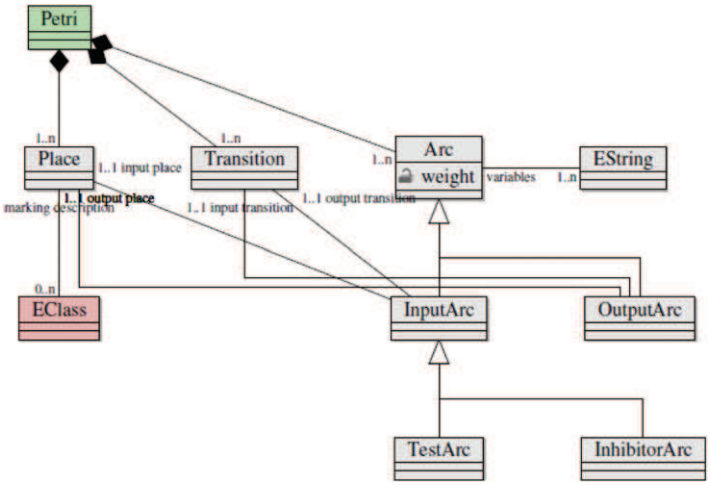


Fig. 2. The Cooperative Objects meta-model.

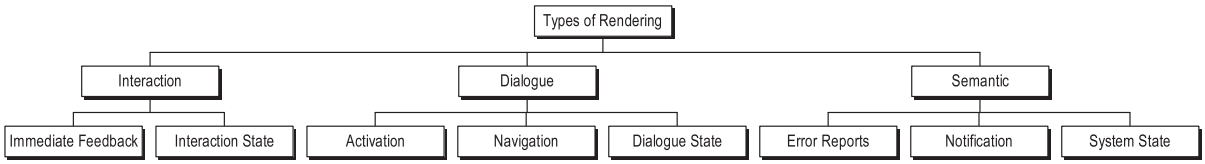


Fig. 3. Hierarchical taxonomy of rendering.

2.2. Rendering limitation of ICO

The presentation part presented in the previous section highlights the fact that ICOs have been initially targeting at WIMP interfaces where interaction takes place through a set of predefined components called widgets. In this kind of interfaces the external appearance is embedded in the widgets themselves and thus independent from the ICO description.

In order to highlight how rendering is described using ICO, we use the structure of the taxonomy of rendering presented in [11] extended in order to incorporate post-WIMP user interface (see Fig. 3).

We use the generic term "rendering" for qualifying any form of communication from the application towards the user. The taxonomy presented here does not attempt to classify the different categories of rendering according to their form or medium, but according to their semantic in the application.

When dealing with post-WIMP interactive applications, an important aspect is the graphical feedback the user is provided with while the interaction takes place. Interaction rendering may be refined into *immediate feedback* (the rendering related to the use of input devices such as drawing the mouse cursor for instance) and *interaction state* (the rendering of the state of a complex interaction sequence such as drawing the ghost of an icon while dragging it, or the animation that shows it has been dropped on the wrong target).

On the dialogue side, the rendering aims at keeping the user informed of the evolution of the interaction that is taking place with the system. For instance this rendering presents which commands are available according to the state of the application. This category can be further refined into *activation rendering* (the rendering of legal actions such as enabling or disabling or widgets), *navigation rendering* (dealing with the navigation among the screens or windows of the application) and *dialogue state rendering* (displaying values that belong to the dialogue component).

On the semantic side, the rendering corresponds to the way the user interacts with semantic objects that belong to the functional core of the system. The user actions may trigger the creation or deletion of objects, obtain information on the states of the objects, change their state or consist of information about success of failure of a request for function execution.

As a UIDL, ICO is able to capture every aspects of rendering according to this taxonomy. The concrete rendering (how graphical elements of the objects change and are effectively rendered) is out of the scope of ICO and is usually abstracted away by a set of rendering methods that belong to the code of the objects. In order to go beyond that level where the entire rendering is defined by programming code, in [6] we investigated the use of a mark-up language (more precisely SVG [57]) to describe the concrete rendering in a more abstract way. This rendering was modifiable both at design time and runtime according to the run-time architecture presented in Fig. 4.

As presented on Fig. 4 this architecture introduces an ad-hoc document object model (DOM) providing a description of the graphical interface and the widgets that compose it. This DOM provides an abstract representation of the presentation

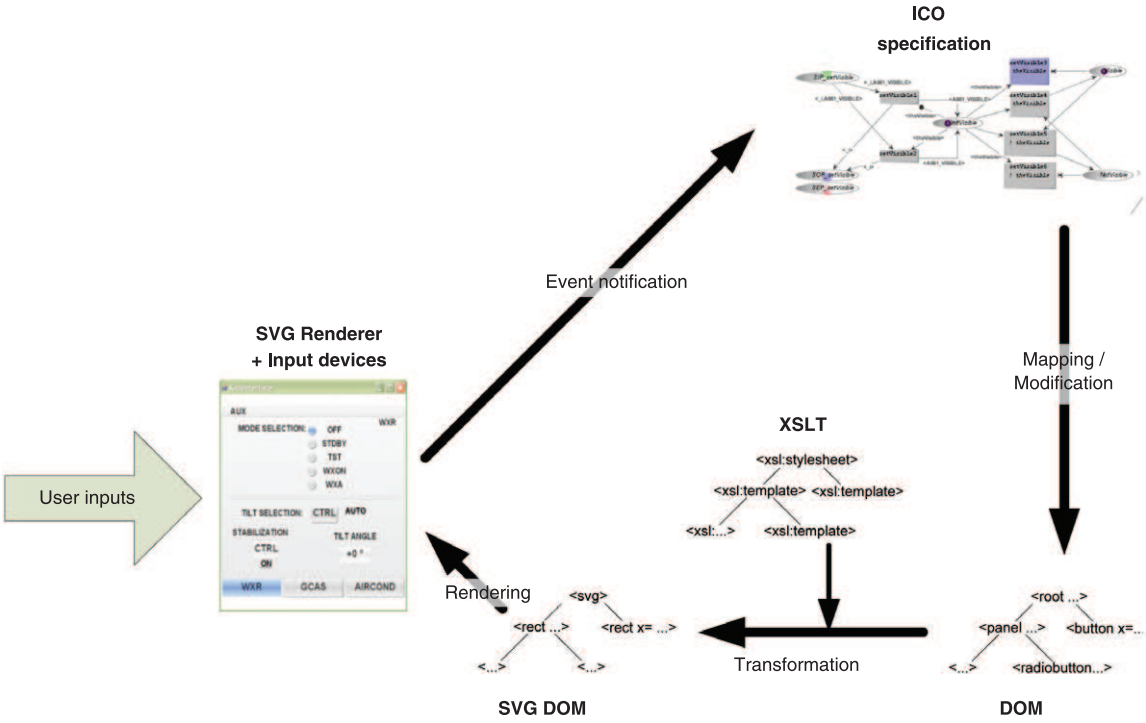


Fig. 4. A run-time architecture using SVG for concrete rendering of ICO.

part of the modelled interactive application modified by the rendering mechanism of ICO. The use of a transformation (an XLST) allows producing a SVG document from the DOM. The SVG document is thus rendered using a dedicated library [67].

This first attempt demonstrated that it is possible to connect the rendering (at an abstract level) to the ICO specification but mainly by using an ad-hoc DOM and its runtime support resulting in degraded performances not suitable for a general purpose UIDL.

2.3. A model-based approach for designing interactive applications

Interactive systems engineering can involve the production of various models such as task model, user model, domain model, dialogue model, training model …that should not be considered independently as they usually co-evolve and represent different views of the same world. Unlike model-driven approaches such as CAMELON [16], the model-based approach promoted with ICOs is not targeting at the derivation/generation/refinement of some models from other ones. On the opposite, the approach promotes the construction of the various models (possibly by different stakeholders involved in the development process) support provided by the approach being more on verifying the consistency between models or identifying mismatches.

As the processes for designing interactive system is by nature iterative (see [42] for a state of the art on such processes) when formal description techniques are used iteration is conditioned by the result of formal verification. This allows proofs to be made on the system model in addition to classical empirical testing once the system has been implemented (the expression and verification of properties has been previously studied for formal notations in the field of interactive systems [21]).

Modelling systems in a formal way helps to deal with issues such as complexity, helps to avoid the need for a human observer to check the models and to write code. It allows reasoning about the models via verification and validation and also to meet three basic requirements notably: reliability (generic and specific properties), efficiency (performance of the system, the user and the two together) and finally to address usability issues (by means of tasks models for instance to assess effectiveness). Fig. 5 presents an example of development process taking into account the integration of system and task models.

Several previous work present how we use this approach to ensure how task and system models are consistent [7] or how training models may be included within this approach [41].

This section used the Arch model to represent rendering coverage in the ICO notation: formal description of the dialogue component, input device management, supporting multimodal interaction, connexion to the functional core and part of the rendering are addressed but ICO suffers a lack of expressiveness when addressing the effective rendering part of the UI. After the review of current approaches in the following section, Sections 5 and 6 present extensions to ICOs in order to overcome this limitation.

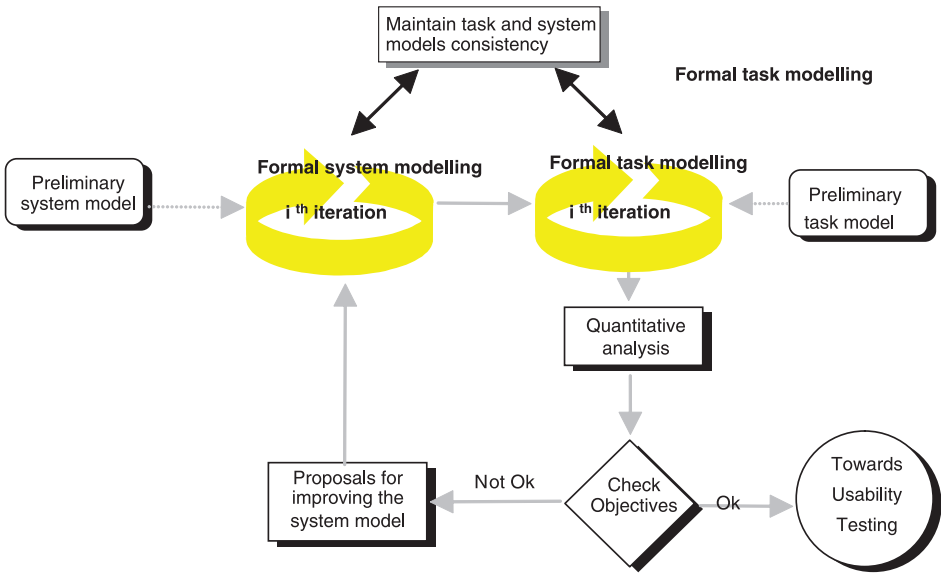


Fig. 5. The iterative model-based design life cycle using both tasks and system models.

3. Related work

This section first presents an analysis of several UIDLs and provides the reader with keys to understand how ICO compares in term of modelling with respect to these UIDLs. We then present an analysis of XML-based languages that could bridge the gap between the description of the abstract rendering described with ICO and the concrete rendering that has to be taken into account in order to make ICOs a full-fledged UIDL. Lastly, based on the two literature overview, we present how it is possible to propose UIDL with means to explicitly take into account both behavioural and rendering aspects of UIs.

3.1. User Interface Description Languages

This section presents a synthesis of the related work studied in [47]. The idea of the original paper was to provide a set of criterion to assess 25 contributions selected either because they are providing a unique and important contribution or because they played a pioneer role in the area of User Interface Description Languages (UIDLs). This study thus allows highlighting the capacity of a UIDL to precisely describe all aspects of both post-WIMP and WIMP user interfaces.

This related work is summarized in Table 1. The UIDLs considered have been gathered (see rows grouping on the left-hand side of Table 1) according to the modelling intrinsic characteristic of the underlying notation or language: **Petri nets-based**, **state-based**, **flow-based, code-based** and **constraints-based**.

In this section, we focus on three types of characteristics for UIDLs (Interaction Coverage, Tool Support and Expressiveness) in order to group and compare previous work. Each characteristic is divided into several sub concerns which are evaluated for each UIDL, showing if it covers the point or not:

- *Y(Yes)* means that characteristic is explicitly handled by the UIDL,
- *N(No)* means that the characteristic is not explicitly handled.
- *Code* means that the characteristic is made explicit but only at the code level and is thus not a construct of the UIDL.

The **Interaction Coverage** aspect presents which components of the architecture of an interactive system are covered by the UIDL, based on the Arch architecture presented in Fig. 1. **Interaction Coverage** is then divided in two different sub-elements: post-WIMP specific elements (such as handling multimodal aspects) and other elements which are implied in both post-WIMP and WIMP interfaces (such as dialogue modelling).

The **Tool Support** aspect defines if websites or papers contain at list snapshots of the application, if a demo of the application is available on a website or if the tool is downloadable.

The **Expressiveness** of the UIDL is presented using six different properties of the language. This expressive power is not a goal per se but it clearly defines the type of User Interface that can be described using the UIDL and the ones that are beyond their reach:

- The first three characteristics deal with description of objects and values in the language (this is named **Data Description**), with the description of states (**State Representation**) and the description of events (**Event Representation**).

Table 1
UIDLs overview.

			Interaction Coverage				Tool support	Expressiveness						
			Post-WIMP specific (Low Level, Multi-modality, Tangible, Fusion)	Other										
Underlying notation	**Language**	**Reference to the first paper**		**Widget**	**Rendering**	**Dialogue**		**Data Description**	**State Representation**	**Event Representation**	**Time quantitative**	**Time Qualitative**	**Concurrent Behaviour**	**Dynamic Instantiation**
Constraints	**Coral**	[58]: Szekely et al. 1988	(N, Y, N, N)	Y	Code	N	N	Code	N	N	N	Y	N	N
	Apogee	[30]: Hudson 1989	(N, N, N, N)	Y	Y	N	Y	Code	N	N	N	N	N	N
Code-based	**Squeak**	[17]: Cardelli et al. 1985	(Y, Y, N, N)	Y	Code	Code	N	Code	N	Code	N	Y	Y	N
	CSP	[56]: Smith et al. 1999	(N, Y, N, Code)	N	Code	N	N	Code	Code	Code	N	Y	Y	N
	DMI	[2]: Anson 1982	(Y, Y, N, N)	N	N	N	Y	Code	Code	Code	N	Y	N	N
	Subarctic	[31]: Hudson et al. 2005	(N, N, N, N)	N	Code	N	Y	N	Code	Code	N	Y	N	N
	XISL	[35]: Katsurada et al. 2003	(N, Y, N, N)	N	Code	Code	Y	Code	N	Code	N	Y	Y	N
	UsiXML	[40]: Limbourg et al. 2004	(N, Y, Y, N)	Y	Code	Code	Y	Code	N	Code	N	Y	Y	N
Flow-based	**GWUIMS**	[53]: Sibert et al. 1986	(N, Y, N, N)	Y	Code	N	Y	Y	N	Code	N	Y	Y	N
	Tatsukawa	[59]: Tatsukawa 1991	(Y, N, N, N)	N	N	N	N	Y	N	Y	N	Y	Y	N
	Marigold	[63]: Willans et al. 2001	(Y, N, N, N)	N	N	N	Y	Y	Y	Y	N	Y	Y	N
	Wizzed	[24]: Esteban et al. 1995	(Y, N, N, Code)	Y	Code	N	Y	Y	N	Y	N	Y	Y	N
	ICON	[22]: Dragicevic et al. 2004	(Y, Y, N, Code)	N	Code	N	Y	Y	N	Y	N	Y	Y	N
State-based	**Swingstates**	[3]: Appert et al. 2006	(N, N, N, N)	Y	Code	N	Y	N	Code	Code	N	Y	N	N
	Hierarchical	[13]: Blanch et al. 2006	(Y, Y, N, Code)	Y	N	N	Y	Code	Code	Code	N	Y	Y	N
	3-State Model	[15]: Buxton 1990	(Y, N, N, N)	N	N	N	N	N	Y	Y	N	Y	N	N
	GTN	[37]: Kieras et al. 1983	(Y, N, N, N)	N	N	N	N	N	Y	Y	N	Y	N	N
	HephaisTK	[23]: Dumas et al. 2008	(Y, Y, Y, Code)	N	Code	Y	Y	Code	Y	Y	N	Y	Y	N
	IOG	[18]: Carr 1994	(Y, N, N, N)	Y	N	N	Y	Y	Y	Y	N	Y	N	N
	Shadow	[33]: Jacob 1986	(Y, Y, Y, N)	Y	Code	Y	Y	Y	Y	Y	N	Y	Y	N
	NiMMiT	[19]: Coninx et al. 2007	(Y, Y, Y, Code)	N	Code	Y	Y	Y	Y	Y	N	Y	Y	N
Petri nets	**Hinckley**	[29]: Hinckley et al. 1998	(Y, Y, N, Y)	N	N	N	N	N	Y	Y	N	Y	Y	N
	OSU	[36]: Keh et al. 1991	(Y, N, N, N)	N	Code	Y	Y	Y	Y	Y	N	Y	Y	N
	MIML	[39]: Latoschik 2002	(Y, Y, Y, Code)	N	Code	Y	Y	Y	Y	Y	N	Y	Y	N
	ICO	This Paper	(Y, Y, N, Y)	Y	Code	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 2
Extended markup languages comparison (the original is from [54]).

Language	Tools support	Runtime compatibility	Behaviour	Introduction of new components
CUI (UsiXML)	Yes	Partial	Set of dedicated tags (partial coverage)	No
MXML (Flex)	Yes	No	Script	Not direct
XAML (Silverlight)	Yes	No	Script	Not direct
HTML5	Yes	No	Script	No
Android XML	Yes	No	Script	Not direct
LZX (OpenLaszlo)	Yes	No	Script	Not direct
FXML (JavaFX)	Yes	Yes	Controller (Java class)	Yes

- **Time** is also an important characteristic for behavioural description of interactive applications. Two different representations of time are addressed here: **Quantitative** and **Qualitative**. Quantitative time represents behavioural temporal evolutions related to a given amount of time (usually expressed in milliseconds). This is necessary for the modelling of the temporal windows in a fusion engine for multimodal interfaces where events from several input devices are fused only if they are produced within a same time frame. Qualitative time aims at representing ordering of actions such as precedence, succession, simultaneity ….
- The last two characteristics are **Concurrent Behaviour** and **Dynamic Instantiation**. Representation of concurrent behaviour is necessary when the interactive systems feature multimodal interactions or can be used simultaneously by several users. Dynamic instantiation of interactive objects is a characteristic required for the description of interfaces where objects are not available at the creation of the interface (as in WIMP interfaces where all the graphical objects are predefined and appear in a window) as for instance in desktop-like interfaces where new icons are created according to user actions. Supporting explicit representation of dynamic instantiation requires the UIDL to be able to explicitly represent unbounded number of states as the new created objects will by definition represent a new state for the system. Most of the time, this characteristic is handled by means of code and remains outside the UIDLs. Only Petri nets-based UIDL can represent explicitly such characteristic provided they handle high-level data structure or objects as this is the case for many dialects [12] or [26].

3.2. Mark-up languages

The goal of the work presented in this paper is to add to ICO modelling capabilities for the effective rendering of user interface. As stated by Table 1, rendering in ICO is currently performed using code. A connection to the ICO editing and execution environment called PetShop [7] to the effective presentation components is done using a dedicated application programming interface (API).

To abstract such mechanism, we studied the possibility of using declarative language. In their paper [54] Carlos Eduardo Silva and José Creissac Campos review a set of markup languages (Flex [25], Silverlight [55], HTML5 [32], Android XML [1] and LZX [50]) to study if they can be used as declarative modelling languages for the user interface and compare them to the UsiXML Concrete User Interface (CUI) model. The result of this study is that such languages are able to express structural aspects of the user interface but provide poor declarative means (usually they use scripting language) to handle behavioural aspects (they underline too that it is the case for UsiXML CUI model which provides a small set of dedicated XML tags).

We summarize in Table 2 the comparison of these languages with a specialized set of criterion, putting a focus on the possible compatibility with ICO:

- Tool support: design-time and runtime are two important aspects of our approach, these two phases requiring tool support. The compatibility is not mandatory at design-time even if it should help in supporting design activity.
- Rendering Compatibility: integration of approaches thus requires at least compatibility of runtime architecture. As ICO runtime architecture is based on Java technology, we identify here if the corresponding language can easily be supported by Java. With UsiXML, this runtime compatibility is partial as the existing interpreters are not ready for coupling with external tools.
- Behaviour: This point highlights only behavioural aspects are handled by the corresponding language.
- Introduction of new components: As stated above, all the studied languages are able to express structured aspects of user interface and cover most of them (use of layout, widgets, event handling, animation …). An important aspect of the ICO approach is that it allows the description of the behaviour of any type of graphical component. To be able to render such component, a language must allow the extension of its own set of graphical components. "Not direct" means that new component cannot be explicitly introduced, but it is possible to use existing mechanisms to provide a graphical representation (vector drawings, graphical composition of components…)

We introduce in this comparison a recent XML-based and Java-based technology called JavaFX FXML which can be used for both rich internet and desktop applications.

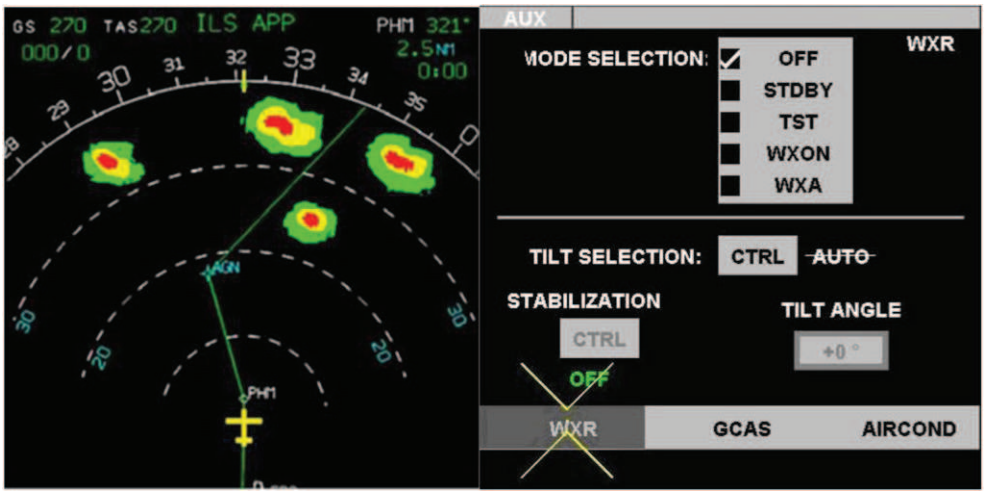


Fig. 6. WXR User Interface of the MPIA application.

3.3. Principles of ICO

The two previous sections present the important aspects a UIDL must cover to fully describe any user interface. As illustrated by Table 1, ICO clearly covers most of the UIDL important aspects, thanks to the Petri net capabilities. Its supporting tool support PetShop allows both editing and execution of models, making it consistent with the model-based life cycle presented by Fig. 5. But, as stated in Section 2.3, an important weakness of this approach is the lack of means to address the effective rendering (as shown by Table 1, the effective rendering is performed using Java code).

Based on the analysis provided by Section 3.2, it is possible to bridge this gap. Table 2 shows FXML is one good candidate to enhance ICO with rendering capabilities, as it covers the three important aspects four our approach (e.g. tool support, possible runtime connection with PetShop and the introduction of new rendering components).

The following section illustrate using a case study how it is possible to integrate these two approaches to provide a first step approach that covers all user interface aspects.

4. Case study

To illustrate the language constructs of our approach, we use the example of an aircraft application called WXR (for Weather Radar System) which is part of the cockpit set of applications. We illustrate also how it is possible to introduce a new component in this application (based on direct manipulation), describing it in a similar way as for the application itself. This case study is also used in the two following sections (Sections 5 and 6) to support the principles of enhancing ICO with means to describe the effective rendering.

4.1. Overview of the MPIA application

To illustrate the language constructs of our approach, we use the example of an aircraft application called WXR (for Weather Radar System) which is part of the cockpit set of applications. It allows the crew members to adjust the orientation (the tilt angle) of the weather radar system when needed (the main idea being to use this feature only if necessary as, most of the time, the default behaviour is the correct one). There are three possible states for tilt angle selection: auto adjustment, auto stabilization and setting up manually the tilt angle (as shown on lower right part of Fig. 6) and five modes of detection independent from the tilt selection (the upper right part of Fig. 6): OFF switches off the weather radar, STDBY switches on the weather radar but the detection is not activated, TST displays graphical test patterns on the radar screen, WXON switches on the weather radar (the warm-up phase should be long before detection starts) and WXA activates weather detection and displays alerts when required. The left part of Fig. 6 is the resulting weather image on the radar screen.

4.2. Modelling WXR page of the MPIA application

Modelling a user application using ICO is quite simple as in the area of interactive cockpits correspond to classical WIMP-based user interfaces. Every widget (*PicturePushButton*, *CheckButtons*, *EditBoxNumeric*) or event (*A661_EVT_SELECTION*, the event raised by a *PicturePushButton*) introduced in this case study comes from the ARINC 661 Specification, which is a standard [4,5] for the interactive aircraft cockpits.

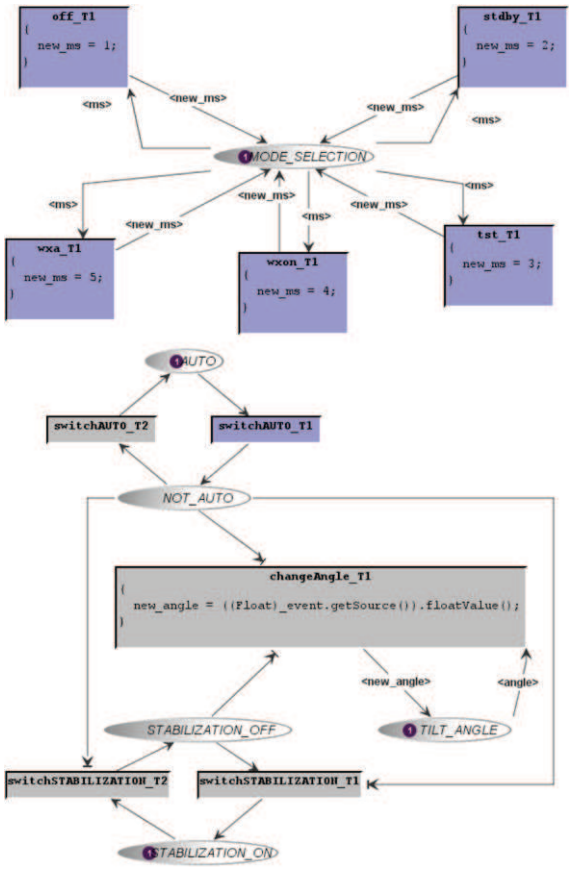


Fig. 7. Behaviour of the page WXR.

Widget	Event	Event Handler
auto_PicturePushButton	A661_EVT_SELECTION	switchAUTO
stab_PicturePushButton	A661_EVT_SELECTION	switchSTABILIZATION
tiltAngle_EditBox	A661_STRING_CHANGE	changeAngle
…		

Fig. 8. Activation function of the page WXR.

Fig. 7 shows the entire behaviour of page WXR which is made up of two non-connected parts:

- The upper part aims at handling events from the 5 *CheckButtons* and the modification implied of the MODE_SELECTION that might be one of five possibilities (OFF, STDBY, TST, WXON, WXA). Value changes of token stored in place *Mode-Selection* are described in the transitions while variables on the incoming and outgoing arcs play the role of formal parameters of the transitions.
- The lower part concerns the handling of events from the 2 *PicturePushButton* and the *EditBoxNumeric*. Interacting with these buttons will change the state of the application, allowing changing the tilt angle of the weather radar.

Fig. 8 shows an excerpt of the activation function for page WXR, which describes the link between events availability and triggering and the behaviour of the application. For instance, the first line represents the link between the event A661_EVT_SELECTION produced by the button *auto_PicturePushButton* and the event handler switch from the behavioural model of WXR (see Fig. 7). If the event handler is available, the corresponding event producer (the button) should be enabled.

From this textual description, we can derive the ICO model as presented in [6]. The use of Petri nets to model the activation function is made possible thanks to the event communication available in the ICO formalism. As this kind of communication is out of the scope of this paper, we do not present the models responsible in the registration of events-handlers needed to allow the communication between behaviour, activation function and widgets.

Fig. 9 shows an excerpt of the rendering function, which describes how state changes within the WXR behaviour lead to rendering changes. For instance, when a token (*<float a>*) enters (i.e. *token_enter*) the place *TILT_ANGLE,* it calls the rendering method *showTiltAngle(a)* which displays the angle value into a text box.

The modelling of the rendering function into Petri nets works the same way as for the activation function, i.e. for each line in the rendering function, there is a pattern to express that in Petri nets (the interested reader may find more details in [6]).

ObCSNode name	ObCS event	Rendering method
MODE_SELECTION	token_enter <int m>	showModeSelection(m)
TILT_ANGLE	token_enter <float a>	showTiltAngle(a)
…		

Fig. 9. Rendering function of the page WXR.

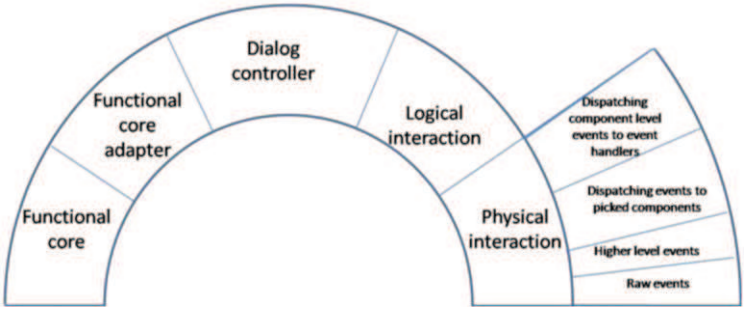


Fig. 10. Refinement of the Arch physical interaction component into four layers (the Arch model is presented in Fig. 1).

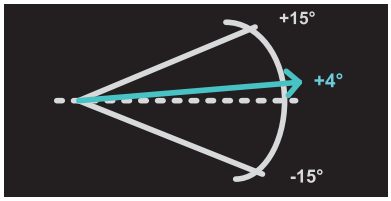


Fig. 11. Graphical component to edit the tilt angle.

For both the activation and the rendering function, the effective rendering is hidden behind a set of methods (for instance *showTiltAngle(float a)* in the rendering function) embedded within a software component that represents the presentation part of the interactive system.

4.3. Creation of a new component (widget) with a dedicated interaction technique

This section presents how the ICO description technique can be used to describe the physical interaction Arch part (right hand side in Fig. 10) of a component. The key point is to describe how raw events coming from the input devices are translated into higher level events used by the logical interaction component. This is a very important aspect for new interfaces as these interfaces typically involve new input or output devices whose events have to be processed and managed within the interactive system.

The various layers represented in Fig. 10 handle events at a different level of abstraction from lower-level events (directly produced by input devices) to dialogue-level (or task-level) events:

- The first level (physical interactions) handles the **raw input events** from the input devices (where the model reads on a regular the state of the device and produces corresponding events).
- The second level contains descriptions which manage **higher level events** (such as clicks, drag …) by combining raw events.
- The third level manages the connection between the higher level events and the graphical components (higher events are used to find the graphical components they are related to, and then the graphical components handle these events according to their inner states to finally produce **widget level events**).
- The fourth level handles the events produced by the graphical components and dispatches them to event handlers (such as the ones generated by User Interface Management Systems).

When dealing with post-WIMP interactive application, it is important to be able to capture each level of event production to ensure the fine grain description of interaction techniques. In our approach, ICO models are used at each level. This architecture has been used for the formal description of many interaction techniques including gesture, voice and multimodal interactions (two-handed, voice and speech …).

As an illustration, we introduce a graphical component for the editing of the tilt angle to replace the edit box in the WXR application (Fig. 11 shows the graphical representation of this component).

This component allows the selection of an angle by dragging the arrow that represents the current angle. As presented, the current angle must remain between the $[-15,+15]$ bounds.

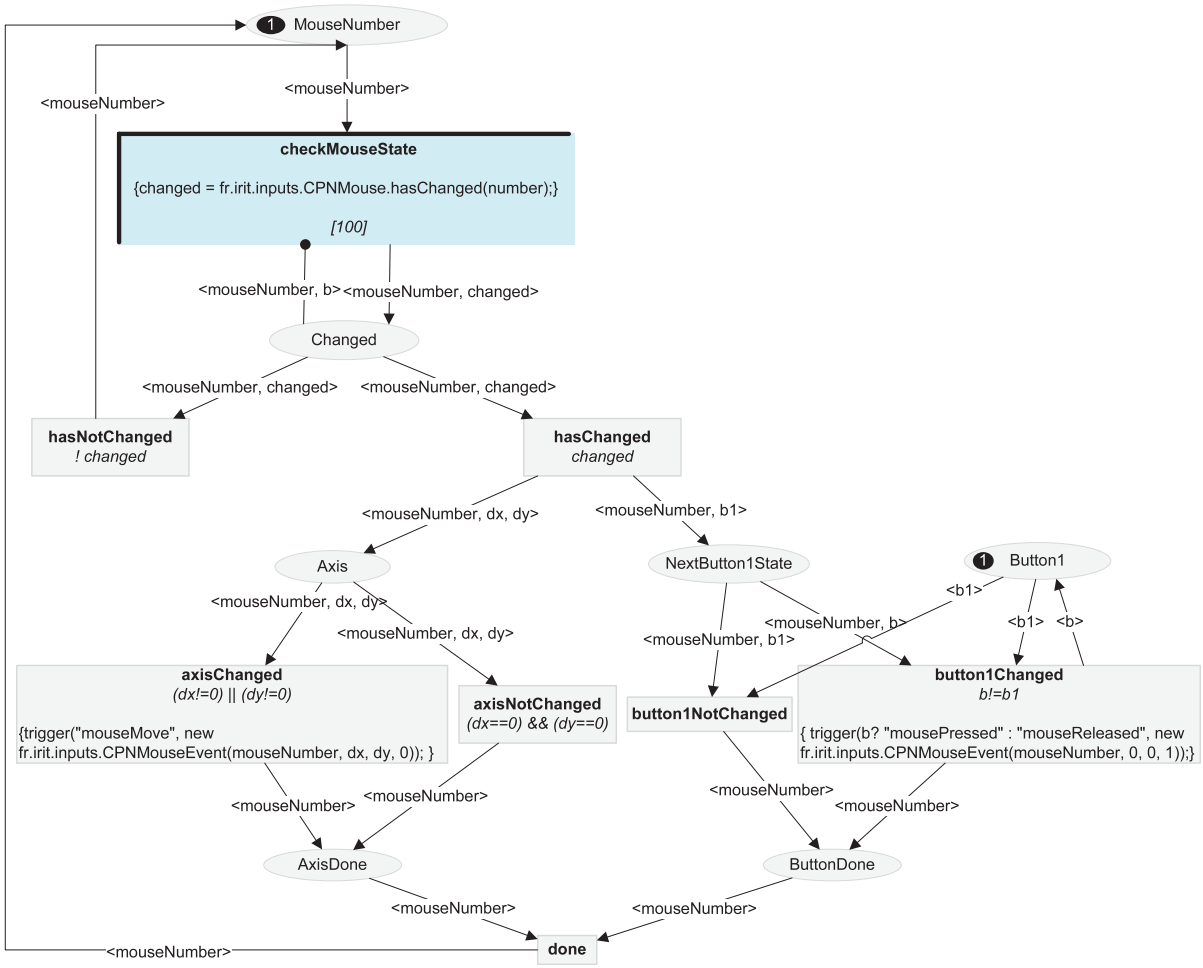


Fig. 12. Model of the raw events handling for both configurations *i*.

4.3.1. Raw events

Fig. 12 presents a model handling raw events from an input device (here the input device is a Keyboard Cursor Control Unit (KCCU) which is available in interactive cockpits and behaves like a mouse) and producing upper level events such as mouseMove, mousePressed, mouseReleased, etc. This model can be read as follows: first the value of a variable "changed" defined in transition *CheckMouseState* (upper transition in Fig. 12) is tested every 100 milliseconds ([100] at the bottom of the transition) in order to verify if the state of the input device has changed since the previous check. According to the value of the variable, transition *hasNotChanged* or *hasChanged* will fire.

Then, two further tests are performed in order to identify the movement of the KCCU and the state of the button. The movement test is represented by transition *axisChanged*, (left hand side of the model) according to (x, y) coordinates (mouseMove). Transition *buttonChanged* (right hand side of the model) checks to see if there has been a change in the state of the KCCU button[1] which produces *mousePressed* or *mouseReleased* events (code "trigger" in the respective transitions). After the *axisChanged* and *buttonChanged* tests, transition *done* is fired placing a token in place *MouseNumber* to be ready to start again the process.

4.3.2. Higher level events

Fig. 13 represents a typical interaction technique that produces simple or double clicks and events corresponding to the dragging activity. The model receives events *mousePressed*, *mouseReleased* and *mouseMoved* from the raw events model presented in Fig. 12 which then processed to raise *click*, *doubleClick* and *beginDrag*, *drag* and *endDrag* events.

The Cooperative Object presented on Fig. 13 behaves as follow:

[1] In the current model the KCCU only features one button.

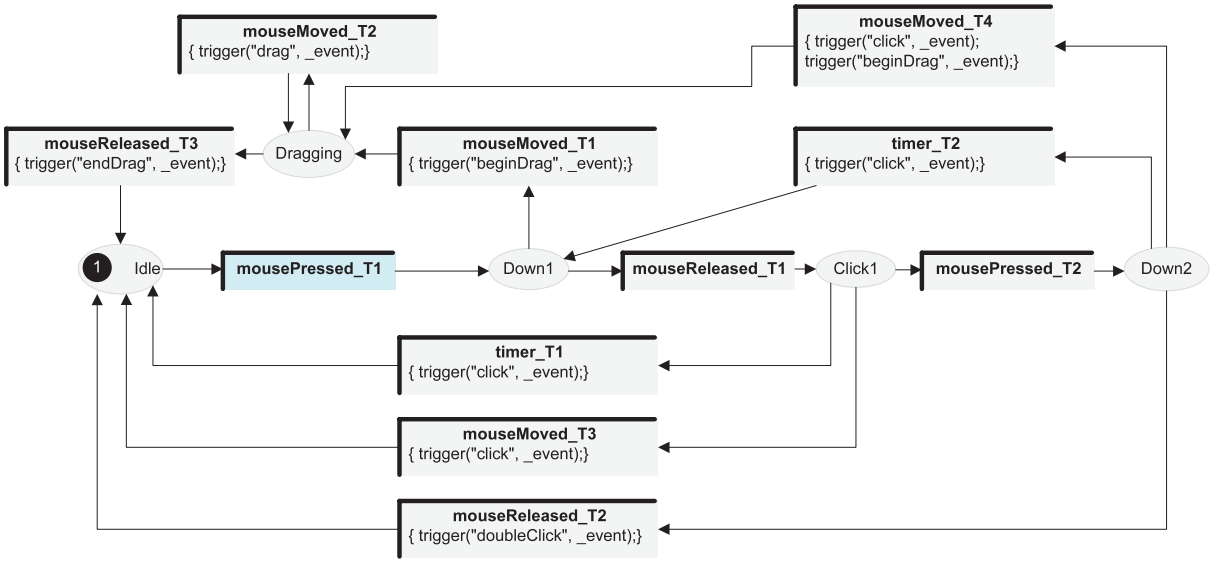


Fig. 13. Example of model producing higher level events *i*.

- The *doubleClick* event occurs when the mouse button has been pressed and released twice within a predefined temporal window and without any *mouseMove* event in-between (firing of transition *mouseReleased_T2* and code trigger raising doubleClick).
- If a *mouseMove* (firing of transition *mouseMove_T3*) or a time out (*timer_T1*) occur after the first sequence of *mousePressed*, *mouseReleased* (firing of transition *mousePressed_T1* and *mouseReleased_T1*) a simple *click* event is produced.
- If a *mouseMove* occurs in between the first sequence of *mousePressed* and *mouseReleased*, a *beginDrag* event is triggered (firing of transition *mouseMove_T1*), and then the model is able to produce *drag* events while receiving *mouseMove* events, or to produce an *endDrag* if the mouse button is released.
- If a *mouseMove* occurs in between the second sequence of *mousePressed* and *mouseReleased*, a simple *click* is triggered at the same time as a *beginDrag* event (firing of transition *mouseMove_T4*). Since then, the model is able to produce *drag* events each time a *mouseMove* event is received, or to produce an *endDrag* event the event *mouseReleased* is received.
- If a timeout occurs in between the second sequence of *mousePressed* and *mouseReleased* (corresponding to the fact that the user has been too slow for performing the second click), a simple *click* is triggered and the model then behaves as if only the first *mousePressed* has occurred (firing of transition *timer_T2*).

4.3.3. Dispatching events to picked graphical components

Dispatching events to graphical components (widgets) requires being able to get these components from the set of graphical components on the user interface. This mechanism is called a "picking mechanism". An example of such mechanism modelled with ICOs is presented in Fig. 14. The modelling principle is here to identify (when an event *mouseMove* is received) which graphical component is under the mouse cursor. If a component is identified, every event (among *mousePressed*, *mouseReleased*, *click*, *doubleClick*, *beginDrag*, *drag* and *endDrag*) is then forwarded to it (possibly resulting in an adaptation of the coordinates). In the current state of Fig. 14 one object has been picked and that object (of which the reference is stored in place MouseOnObject) is ready to receive the set of events (mouseMove, mouseClick, mousePressed, mouseReleased and beginDrag) as the corresponding transitions are enabled (graphically represented in dark grey).

The graphical component able to receive these events must handle them. This handling might have an impact on the inner state of the component as well as on its graphical appearance. To describe such graphical components, we use part of the ICO notation: a cooperative object, a logical presentation part and a rendering function. There is no need for an activation function as there is no deactivation of the events production).

Fig. 15 shows the set of methods used to render the state changes of the angle selector component (*showDraggingArrow* which is used to render the dragging of the arrow showing the selected angle; *showArrow* which renders the current value of the angle by drawing the arrow; *showEnabled* which renders the component is enabled or not depending on the value of its parameter).

Fig. 16 presents the behaviour of the angle selector:

- The upper part defines whether the component is enabled or not according to the service call of *setEnabled*. A token is placed in the place *Enabled* or *NotEnabled* depending on the value of the parameter of the method call (that can be TRUE or FALSE).

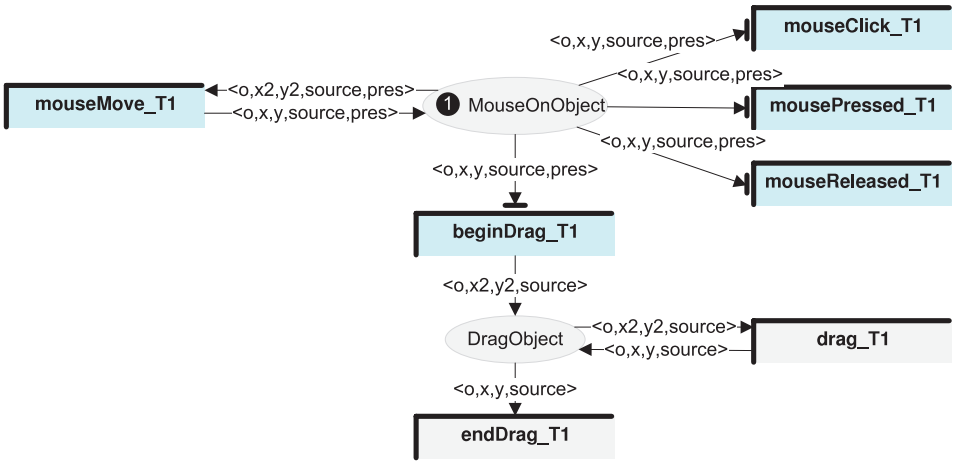


Fig. 14. Example of a picking manager *i*.

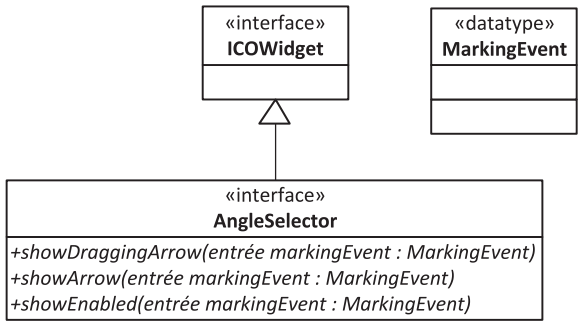


Fig. 15. Software interface of presentation part of the angle selector component.

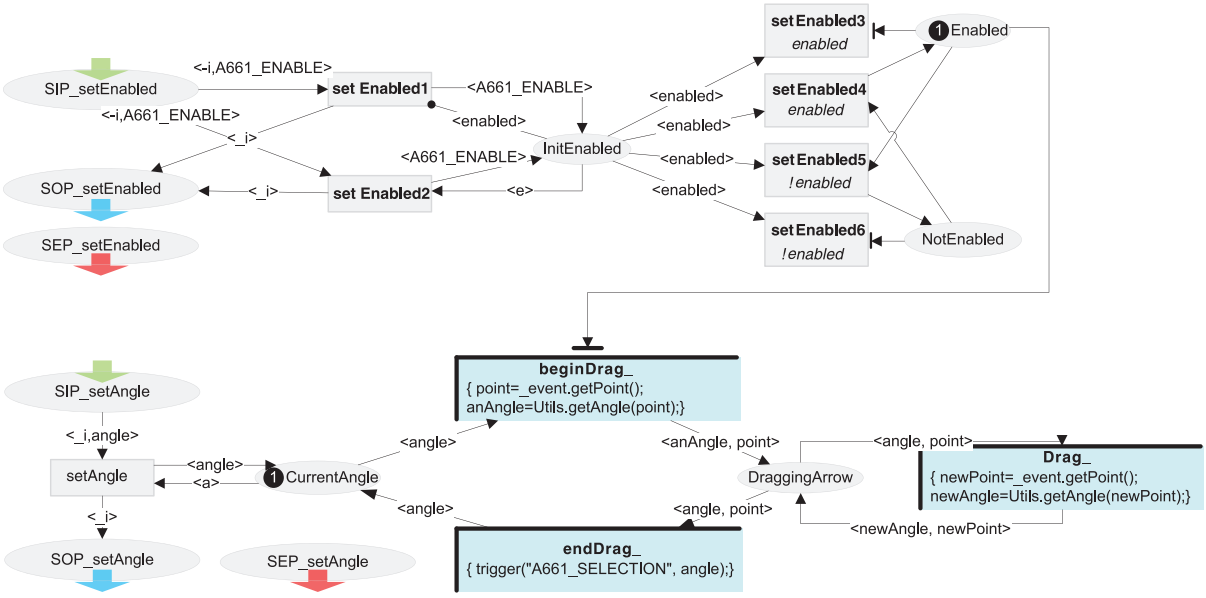


Fig. 16. Behaviour of the angle selector component *i*.

- The lower part handles the dragging of the arrow. The token in place *CurrentAngle* holds the current value of the angle. If the component is enabled and the event *beginDrag* is received, the token is moved to place *DraggingArrow* which is updated for every occurrence of a *drag* event. The code inside the transitions *beginDrag_* and *drag_* corresponds to retrieve the mouse cursor coordinate from the event received, and compute through a dedicated method the value of

ObCS Node name	ObCS event	Rendering method
CurrentAngle	token_enter	showArrow
DraggingArrow	token_enter	showDraggingArrow
Enabled	token_enter	showEnabled
Enabled	token_remove	showEnabled

Fig. 17. Rendering function of the angle selector component.

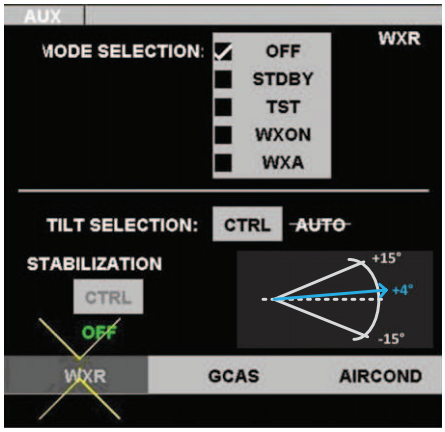


Fig. 18. Integration of the angle selector within the WXR application.

the corresponding angle. When the *endDrag* event is received, the component triggers the event called *A661_SELECTION*[2] that is managed by the rest of the application.

Fig. 17 presents the rendering function which relates state changes to rendering. For instance, the first line describes that when a *token enters* the place *CurrentAngle*, the method *showArrow* is called with the value of the angle (hold by the token) as a parameter (see Fig. 15).

With these 3 elements the angle selector component is fully described, and may be used within a window as described in next section.

4.3.4. Dispatching component level events to event handlers

Fig. 18 shows the graphical integration of the angle selector component within the WXR application. Beyond the graphical integration, two main connexions have to be dealt with:

- Handling the event produced by the component (called *A661_SELECTION*)
- Linking the event handling with the event production at the logical interaction level (production of the event *asked_changeAngle*).

This integration depends on the technology used to implement the physical interaction part. For instance, using Java language, the event handler will be represented by a dedicated listener (*angleSelectorListener*) which will receive the notification of the *A661_SELECTION* and then throw the call of a dedicated method (*void A661_SELECTION(A661SelectionEvent anEvent)*). Then, in this method, some code would be dedicated sending a notification to the logical interaction level with a line such as *getPhysicalPart().notifyEvent("asked_changeAngle", someValue)*. Another possibility would be to model a cooperative object with a similar behaviour.

This example shows how the ICO can be used not only for modelling interactive applications but also for modelling an interaction technique. These two elements have been used as running examples for presenting the basic characteristics of the ICO.

5. Integration of ICO with FXML, a markup language for GUI description

Within the CNES (*Centre National d'Etudes Spatiales*, a.k.a. French National Space Studies Center) Research & Technology project called ALDABRA, we investigate the use of JavaFX technology as a mean for describing the graphical part of a user interface which dynamic behaviour is modelled using ICO. The requirement from our industrial partner was to use a Java technology supported by an important company and that can be handled within classical integrated development environment such as NetBeans and Eclipse. The choice of Oracle JavaFX was thus natural as it allows the description of user interfaces using an XML based language called FXML.

2 This name in defined in ARINC 661 specification standard [ARINC 661, 2005].

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  …
3.
4.  <AnchorPane id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-
    Infinity" prefHeight="400.0" prefWidth="316.0" xmlns:fx="http://javafx.com/fxml"
    fx:controller="fr.irit.aldabra.mpia.ICOcontroller">
5.  <children>
6.  …
7.  <VBox id="TILTEdition" alignment="CENTER" prefHeight="80.0" prefWidth="89.0" spacing="10.0"
    AnchorPane.bottomAnchor="0.0" AnchorPane.rightAnchor="22.0" AnchorPane.topAnchor="0.0">
8.  <children>
9.  <Label id="label7" text="TILT ANGLE" />
10. <TextField id="angleEditor" alignment="CENTER" onAction="changeAngle" prefWidth="82.0" text="+0°" />
11. </children>
12. </VBox>
13. …
14. </children>
15. <stylesheets>
16. <URL value="@defaultMPIAStyleSheet.css" />
17. </stylesheets>
18. </AnchorPane>
```

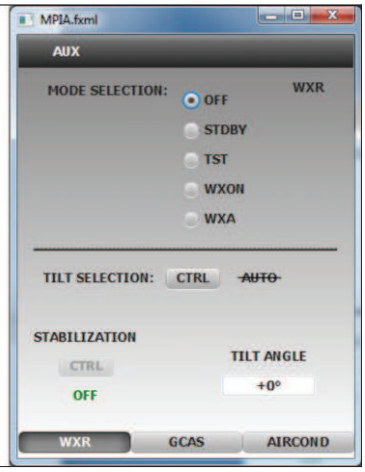


Fig. 19. FXML description of the MPIA graphical part.

In this section, after a brief recall of the main JavaFX features, we present an approach to couple FXML with ICO, using the same case study introduced in the previous section.

5.1. JavaFX and FXML principles

JavaFX 2.0 (http://javafx.com) is a Java technology developed to support the implementation of Rich Internet Applications. As it is compatible with the two main Java graphical toolkits Java Swing and IBM SWT, it allows the implementation of 2D application with complex rendering, including animations.

JavaFX supports the use of an XML based language called FXML to describe the graphical user interface, where interactive objects and their layout are handled by XML tags, while the behaviour and the connexion to the functional core are handled using Java technologies. Styling is ensured by the use of CSS3+ style sheets. There exist more than 50 kinds of widgets that may be used for any application, including classical widgets and more complex one such as tables, editors… It is also possible to include new widget classes that automatically extend the set of FXML tags.

JavaFX is fully supported by Oracle who provides tools to design application, such as a graphical editor called Oracle Scene Builder, plugins within the integrated development environments NetBeans™ and Eclipse™ for generating code or JavaFX components from SVG description.

The next sections illustrate the JavaFX FXML mechanisms and how it is possible to integrate ICO and the FXML using the previous example of MPIA.

5.2. Modelling MPIA using ICO and FXML

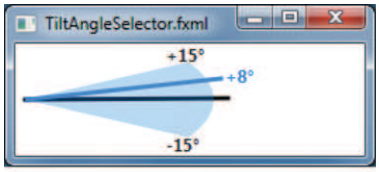
Using the FXML language, it is possible to fully describe the MPIA graphical part. Fig. 19 illustrates both the FXML and the resulting layout for the application. The FXML excerpt only shows the subpart describing the angle selection and the whole description may be found in Annexe A.

The xml text within Fig. 19 allows the definition of four important aspects:

- The first part is the definition of the widgets of the application and their layout. Any widget is instantiated following the description provided within the xml file (for instance, a text field is created following the description at line 10 in Fig. 19). The layout is ensured by the type of container the widget belongs to and by constraints that describe how the widget is contained relatively to the container. For instance, the top container of this application is an AnchorPane (line 4), allowing putting widget relatively to its borders. The VBox (another container that embeds both the label and the text field for the angle selection) is placed relatively to the bottom, right and top borders of its containing AnchorPane (defined line 7, by the constraints *AnchorPane.bottomAnchor="0.0" AnchorPane.rightAnchor="22.0" AnchorPane.topAnchor="0.0"*).
- A second important part is the possibility of describing a controller that is a Java class, instantiated with the JavaFX window. When making the integration with ICO, the controller is used at runtime to make the link between the behaviour of the application described using ICO and the JavaFX graphical part. In the example, the controller is defined line 3 (*fx:controller="fr.irit.aldabra.mpia.ICOcontroller"*). This controller, using JavaFX mechanisms is able to modify any attributes of the widgets within the application (any widget with the attribute *fx:id* defined is directly map to an attribute of the controller), and then to implement the rendering methods required by the rendering function defined by Fig. 9.
- A third aspect is the used of style sheets to define rendering attributes of the graphical scene. It may define fonts, colours, shapes …of any widget. In the example, the style sheet is defined line 16.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  …
3.
4.  <AnchorPane id="AnchorPane" onMouseDragged="drag" onMousePressed="beginDrag"
    onMouseReleased="endDrag" maxHeight="80.0" maxWidth="-Infinity" minHeight="-
    Infinity" prefHeight="80.0" prefWidth="183.0" xmlns:fx="http://javafx.com/fxml"
    fx:controller="fr.irit.aldabra.mpia.ICOAngleController">
5.    <children>
6.      <Line id="baseLine" endX="125.0" endY="-1.0" layoutX="26.0" layoutY="40.0" startX="-19.0"
    strokeWidth="3.0" />
7.      <Line fx:id="arrow" endX="50.39484786987305" endY="1.0253764390945435" layoutX="96.0"
    layoutY="30.0" rotate="352.0" startX="-88.0" startY="4.0" stroke="DODGERBLUE" strokeWidth="3.0" />
8.      <Arc id="arc1" fill="#1e90ff53" layoutX="24.0" layoutY="40.0" length="60.0" radiusX="100.0"
    radiusY="100.0" scaleX="1.3412279806726912" scaleY="0.5369465863735433" startAngle="330.0"
    stroke="BLACK" strokeType="INSIDE" strokeWidth="0.0" type="ROUND" />
        <Label fx:id="angleValue" layoutX="149.0" layoutY="15.0" text="+8°" />
9.      <Label id="label1" layoutX="107.0" text="+15°" />
10.     <Label id="label2" layoutX="107.0" layoutY="64.0" text="-15°" />
11.   </children>
12.   …
13.   </AnchorPane>
```

ObCS Node name	ObCS event	Rendering method
CurrentAngle	token_enter <x>	showArrow: rotate=360-x
DraggingArrow	token_enter <x>	showDraggingArrow rotate=360-x
Enabled	token_enter	showEnabled styleClass="enabled"
Enabled	token_remove	showEnabled styleClass="disabled"




```
1.
2.  <?xml version="1.0" encoding="UTF-8"?>
3.
4.  <AnchorPane id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
    minWidth="-Infinity" prefHeight="400.0" prefWidth="316.0" xmlns:fx="http://javafx.com/fxml">
5.    <children>
6.    …
7.      <fx:include fx:id="TILTEdfition" source="TiltAngleSelector.fxml" AnchorPane.bottomAnchor="0.0"
    AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
8.    …
9.    </children>
10.   …
11.   </AnchorPane>
```

Fig. 20. Introduction of a new component within MPIA.

- Lastly, it is possible within FXML to handle event, using as a callback a public method of the controller. In the example, the parameter *onAction="changeAngle"* of the text field (line 10) maps the event triggered by the use of the text field with the method changeAngle of the controller, producing the event required by the activation function of Fig. 8.

The integration of ICO and FXML is based on the controller object. The controller plays the role of an adapter that on one side forwards the event produced by the JavaFX components to the activation function and on the other side allows any ICO to invoke methods that are translated into property modification within each component. To do that, in the ICO models of both the rendering and the activation functions, a token may hold a reference to this controller allowing the model to ask for the modification of any property of the graphical part of the application.

5.3. Introduction of the angle selector component

JavaFX make it easy to introduce new components: any application may be used as a node and introduced within any other application, and it may define new properties or events. A single FXML tag (*fx:include*) allows this introduction of new components and the customization of their properties, allowing the creation of an ICO–FXML application that will be integrated within another application as a component.

The upper part of Fig. 20 shows the description using FXML of the angle selector introduced in Section 2.4. It precisely shows how it is made up with graphical lines, arc …and that it provides its own controller with event notifications when

dragging (bolded text of line 4). It is integrated with ICO in the same way as in the previous example (using the controller) to produce the component and its behaviour introduced in Section 4.3. The rendering function remains the same as described by Fig. 17 with more details on how the rendering is performed in terms of attributes modification (for instance the rotation of the line showing the selected angle) or in terms of style, defined in the style sheet of the application (selection of the style "enabled" when the component is enabled). In the current integration of ICO and FXML, the described rendering is implemented using Java code within the controller.

The lower part of Fig. 20 shows how it is possible to easily introduce the new component within the application using the dedicated tag *fx:include* (line 7 of the xml text). At runtime, this tag forces the instantiation of the JavaFX component and of its controller, making the link with the ICO model.

5.4. Limitation of the coupling ICO–FXML

The integration of ICO and JavaFX has been experimented on several case studies, making the description of the corresponding application efficient both at design time and at runtime providing a high quality rendering including animations. However, there is still a gap to bridge as part of the rendering mechanism is embedded within the definition of the controller. This controller still hides in Java code some rendering mechanisms that are difficult to abstract away. Merging this approach with the work done previously in SVG and presented above [20] may overcome this problem by the introduction of transformation of models, but this part of the work is still in progress and could be addressed by an approach based on UsiXML.

Next section presents this integration of ICOs with UsiXML which consist in a powerful alternative to ICO–JavaFX integration.

6. Integration of ICO with UsiXML

While JavaFX presents interesting properties to make ICO able to fully describe most of the interactive systems, integrating ICO and UsiXML appears in a long term view to be a solution to cover most of the description of an interactive system. As mentioned in the introduction of this article, UsiXML is as an emergent standard for describing interactive system and its supporting design process allows handling properties for new interactive systems such as context awareness, cross-platform adaptability… As it is widely supported by the HCI community it guarantees the scientific basis of concepts that are outside of ICO but complementary. Conversely, UsiXML proposes a poor coverage of behavioural aspects and the integration with ICO could overcome this flaw.

The following section proposes an overview of how UsiXML handles behavioural aspects and the next sections present a bridge between ICO and UsiXML and its advantages, based on the work that has already been done on the WXR application and JavaFX.

6.1. UsiXML and behavioural descriptions

UsiXML (USer Interface eXtensible Markup Language) is defined in a set of XML schemas where each schema corresponds to one of the models containing attributes and relationships in the scope of the language [40]. UsiXML schemas are used to describe at a high level of abstraction the constituting elements of the UI of an application including: widgets, controls, containers, modalities, interaction techniques, etc. The UsiXML language is structured according to the four levels of abstractions as proposed by the framework Cameleon [16], as follows: *task models, abstract user interfaces (AUI), concrete user interface (CUI)* and *final user interface (FUI)*. Several tools [44] exist for editing specification using UsiXML at different levels of abstraction. Notwithstanding, developers can start using UsiXML schemas at the abstraction level that better suits their purposes.

As far as the behaviour is a concern, there are some dedicated schemas in UsiXML. At the task level, behaviour is covered by task models featuring operators in a similar way as it is done by CTT [43]. At the AUI and CUI levels several schemas allows to describe basic elements of the dialogue behaviour including *events, triggers, conditions, and source* and *target* components. These elements can be refined at the FUI level to reach final constructs implemented by the target platform.

So far there is limited support for UsiXML schemas related to behavioural aspect of interactive systems beyond the task model level. Some extensions have been proposed to describe high level dialogue behaviours such as those implemented by transitions between windows [62] and between states of workflow-based applications [28]. However, all these extensions are more or less related to task models. The description of fine-grained behaviour in UsiXML is awkward as the behavioural aspect and the user interface composition are interleaved in a single description. So that, the description of events, triggers and actions is scattered along the components of the user interface with makes extremely difficult to visualize the behaviour of the current state of the application being modelled. Another conceptual issue with dialogue modelling with UsiXML is related to the different levels of abstraction; whilst abstract containers can be easily mapped to windows, it is not so easy to envisage abstract behaviour and how to refine them into more concrete actions on the user interface.

A few works [51,64] have addressed the behaviour aspect of interactive system described with UsiXML. Schaefer, Bleul, and Mueller (2006) [51], propose an extension of UsiXML by the means of a dedicated language called *Dialog and Interface*

```
<cuiModel id="WXR-cui_1" name="WXR-cui">
<window id="window_component_0" name="window_component_0" width="456" height="416">
<inputText id="txt_tiltAngle" name="txt_tiltAngle" isVisible="true" isEnabled="true" textColor="#000000" maxLength="50" numberOfColumns="15" isEditable="true"
text=""/>
<button id="btn_switchAUTO" name="btn_switchAUTO" isVisible="true" isEnabled="true" textColor="#000000">
<behavior>
<event id="switchAUTO" eventType="action" eventContext=""/>
</behavior>
</button>
…
</window>
</cuiModel>
```

Fig. 21. Part of the CUI model of the WXR application.

Widget	Event	Event Handler
btn_switchAUTO	A661_EVT_SELECTION	switchAUTO
…		

Fig. 22. Activation function of the page WXR.

ObCSNode name	ObCS event	CUI attribute
TILT_ANGLE	token_enter <float a>	"text" of txt_tiltAngle
…		

Fig. 23. Rendering function of the page WXR.

Specification Language (DISL). The main contribution of that work is to propose clear separation between presentation, user interface composition and dialogue parts of the interface. Winckler et al. (2008) [64] suggest there is no need of new dialogue language as UsiXML can be coupled with existing dialogue modelling techniques such as StateWebCharts (SWC) [65] to deal with the behaviour of interactive systems. Those authors propose a set of mappings that allows SWC specification to be used as running engine for the behaviour of UsiXML specifications. Notwithstanding, the work was limited to navigation between web pages in Web-based user interfaces.

6.2. A proposal for concrete behavioural description within UsiXML

Beyond the obvious link that exists between the domain model of UsiXML and the behavioural description of an ICO, the work presented in the previous sections shows that there are common concerns between UsiXML CUI model and the work about the SVG rendering with ICO in [6] (such as description of high level widgets and graphical interface, independent from implementation), and it shows that it is possible to enhance such descriptions with behavioural aspects.

Based on the work presented in the previous sections, we propose here possible cooperation between UsiXML and ICO, making possible to ease the design path from the concrete user interface to the final user interface.

6.2.1. Introducing behaviour at CUI level

Mapping state changes described using ICO description technique with UsiXML model attributes can be done easily. We illustrate the principle of introducing behavioural aspects at the CUI level with the example of the WXR application. These illustrations provide the key features allowing integration of ICO and UsiXML.

Fig. 21 introduces a subpart of the CUI model of the WXR application, showing only a classical text box and a button:

- The *inputText* element *txt_tiltAngle* aims at containing a number representing a tilt angle.
- When clicked, the button *btn_switchAUTO* produces an event "*switchAUTO*". Both the availability of this event and its occurrence are related to the behaviour of the application (as stated by the next paragraphs).

Making the link between the behaviour of the application expressed using ICO (as illustrated by Fig. 7) is quite easy as there can be a direct mapping of the event produced by the button ("*switchAUTO*") and the available event handler of the behaviour of WXR ("*switchAUTO*"), as shown by Fig. 22.

When the event handler is enabled, the attribute enabled of the button is thus set to *"true"*, *"false"* otherwise.

Describing the rendering of the application is linked to attribute modification of the CUI DOM such as described by Fig. 23.

When the token enters the place *TILT_ANGLE*, the attribute "text" of the *inputText* element of the CUI is modified with the value hold by the token.

6.2.2. An executable CUI as a prototype for FUI

Thanks to the possibility of interpreting and executing Petri nets, ICO's supporting software development environment, PetShop, enables prototype modification at runtime of an application [46]. For instance, the WXR application has been fully

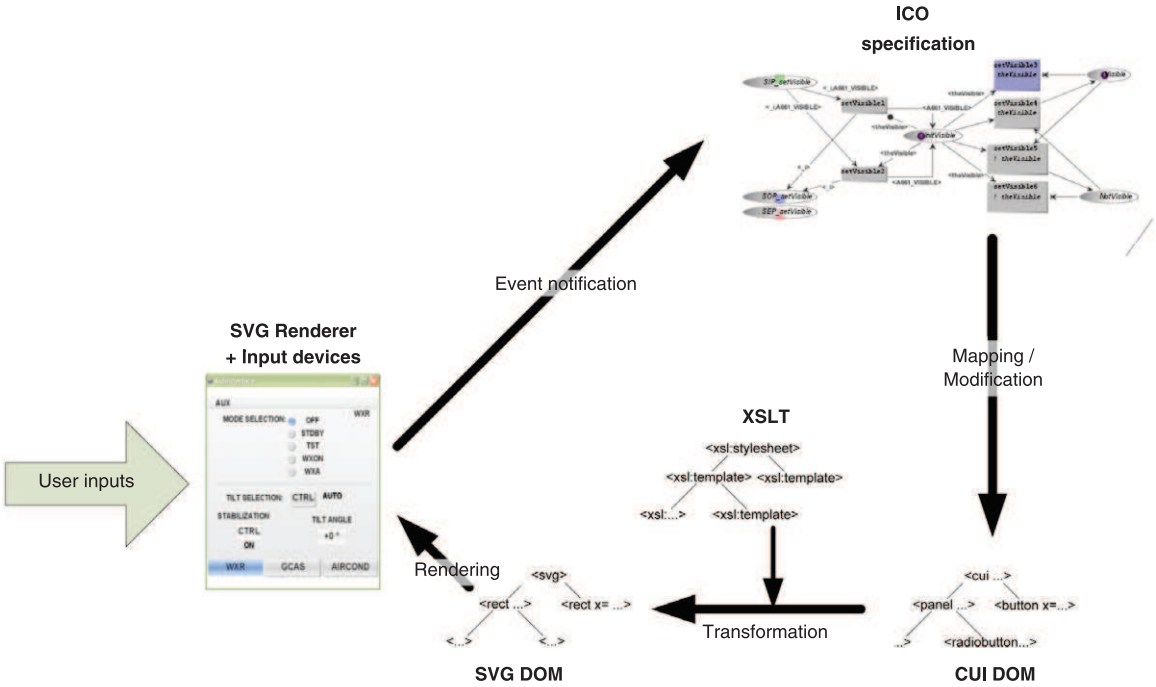


Fig. 24. The run-time architecture.

modelled and can be executed on the CDS modelled using the ICO formalism. However, it has also been connected on a CDS developed on an experimental test bench.

Providing a graphical representation of the CUI makes possible to build a prototype based our approach. Associating ICO and the CUI model has been discussed in the previous section, but it is possible too, in a similar way, to do this association at widget level, while proposing a way to render a CUI model based on a previous work integrating SVG [6], or on the work done using JavaFX. With JavaFX, the approach could take advantage of the possible mapping that exists between the CUI tags and the FXML tags.

Such a work should then allow the prototyping of the final UI (FUI) based on the bridge between ICO and a CUI model, shortening the design path to the FUI.

6.2.3. Rendering based on SVG

As stated when discussing the architecture of Fig. 4, it is possible to clearly separate behavioural aspects from rendering aspects. Any state change of the application should be rendered via the modification of the CUI DOM, based on the mapping described by both the rendering and activation function. Fig. 24 illustrates the run-time architecture that supports FUI prototyping based on the association of UsiXML and ICO.

To provide a rendering to each CUI element, we propose the use of declarative descriptions of the graphical part that support transformations from conceptual models to graphical representations. The approach exploits both the SVG language [57] for graphical representation, and the XSLT language for transformation (called a "*stylesheet*").

In order to write a stylesheet, one has to design the rendering of a particular widget, using Illustrator for example. When ready, the textual description of the widget is included in the stylesheet.

In our case, the source is the CUI DOM, built at start-up time, together with the instantiation of the ICOs components. Before running the application, the system must compile the stylesheet to an XSLT transformer. While running the application, every time the state of a CUI DOM variable changes, it is transformed into a DOM SVG tree, which in turn is passed to the SVG renderer and displayed.

6.2.4. Introduction of behaviour for widgets

To go further with a precise prototyping of the FUI, it is necessary to describe each widget, including its behaviour (such as already done with ARINC 661) [45] . As illustrated by Figs. 16 and 17, it is possible to describe the fine grain behaviour of a widget and the link of its inner state changes with rendering.

In its current state, UsiXML, via the CUI model, describes widgets as a type and a set of attributes (a button is defined by an id, a name …), making it abstract enough to be independent from the targeted platform for the FUI. But when considering prototyping, it may be interesting to provide a finer description of the kind of widget that is expected, and a less coarse grain description of the widgets attributes (for instance, it is possible to introduce rendering for any inner state of a button:

armed, pressed …). Another interesting point when dealing with widget is the introduction of new widgets that may request a precise description of how it should work on the targeted platforms.

One possible way to allow such description within UsiXML could be to enhance the current platform model of the context model with a precise widget description. Even if no effort has already been put on it, this way is an important part of our future works.

7. Discussion and perspectives

This paper has presented a bridge between an already existing formal description technique for behavioural aspects of interactive systems and an approach for describing the presentation part of these systems. Furthermore, this paper highlights how it is possible to take advantage from both behavioural and mark-up language description techniques to propose a new model-based approach for prototyping and developing interactive systems. More precisely, we have presented how it is possible to extend ICO with a mark-up language in order to provide support for description of concrete rendering. The originality of the bridge between ICO and UsiXML lies in the fact that most of the recent work on UsiXML have been focused on mapping UsiXML schemas between several levels of abstraction (for instance [43,60]) or proving automatic user interface generation of components to multi-target devices (as in [44,61]), while very little work have focused on the behavioural aspect of interactive systems modelled with UsiXML.

We systematically demonstrated (using the Arch architecture as a structuring concept) that this approach allows a clear separation between graphical aspects, behavioural aspects and functional aspects. This approach also provides support to distinguish precisely interaction aspects from user tasks (as already argued in previous work [7]). Such a separation is necessary as, depending on the functional part of the interactive system, constraints which are independent from task concerns may appear preventing from the possibility of generating interaction from a description of tasks. Indeed, in the case study example presented in this paper, the value of the tilt angle must meet the system requirements and the dialogue is thus specially designed to support this constraint. If the functional part changes, the dialogue part must be modified, but the user's tasks might remain the same.

It is noteworthy that the use of ICO models to describe the behaviour of user interfaces allows overcoming some of the limitations of other UIDL languages such as SCXML [66] or XUL [68]. It enables managing UIs with infinite states and encapsulation of variables as objects of any kind and dynamic instantiation of objects. Moreover, properties of UI descriptions can be formally assessed using the underlying Petri Net formalism.

However, several points have still to be investigated and four ways of improvement for this work are currently investigated:

- A possible extension to introduce a notation or to enhance the current context model of UsiXML with a precise widget description, including its behaviour, making possible to build prototypes of the Final User Interface.
- An extensive analysis of the use of this approach to precisely describe new types of interactive components (post-WIMP, multimodal …) for which only the ICO approach has been applied to.
- A dynamic connexion between task models/abstract UI and concrete UI, based on a correspondence framework to connect task and system models [7].
- At the tool support level, we currently investigate how to take advantage from the work already done with FXML and SVG to build an equivalent approach using UsiXML, supporting the proposal of Section 6.2.

These extensions are investigated in two industrial projects; the first one deals with tactile interactions in future cockpits of large civil aircrafts while the second focuses on interfaces of satellite ground segments.

Acknowledgements

This work is supported by the Research & Technology Project (RT) ALDABRA (Architecture and Language for Dynamic and Behaviourally Rich interactive Application) (IRIT-CNES).

Annexe A. FXML code of the MPIA graphical part

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>
<?import javafx.scene.shape.*?>

<AnchorPane id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
```

```
prefHeight="400.0" prefWidth="316.0" xmlns:fx="http://javafx.com/fxml" fx:controller="fr.irit.aldabra.mpia.
ICOcontroller">
  <children>
    <GridPane id="gridPane1" prefWidth="599.9999000000025" AnchorPane.bottomAnchor="0.0"
    AnchorPane.leftAnchor="1.0" AnchorPane.rightAnchor="1.0">
      <children>
        <ToggleButton id="toggleButton1" prefWidth="200.0" selected="true" text="WXR"
        GridPane.columnIndex="0" GridPane.rowIndex="0" />
        <ToggleButton id="toggleButton2" prefWidth="200.0" text="GCAS" GridPane.columnIndex="1"
        GridPane.rowIndex="0" />
        <ToggleButton id="toggleButton3" prefWidth="200.0" text="AIRCOND" GridPane.columnIndex="2"
        GridPane.rowIndex="0" />
      </children>
      <columnConstraints>
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
      </columnConstraints>
      <rowConstraints>
        <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      </rowConstraints>
    </GridPane>
    <AnchorPane id="header" prefHeight="31.0" prefWidth="598.0" AnchorPane.leftAnchor="1.0"
    AnchorPane.rightAnchor="1.0" AnchorPane.topAnchor="0.0">
      <children>
       <Label id="lblAUX" alignment="CENTER" prefHeight="29.0" prefWidth="89.0" text="AUX" textAlignment="CENTER"
        AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.topAnchor="0.0" />
      </children>
    </AnchorPane>
    <Separator id="separator" layoutY="201.0" prefWidth="569.0" AnchorPane.leftAnchor="15.0"
    AnchorPane.rightAnchor="16.0" />
    <AnchorPane id="anchorPane1" prefHeight="169.0" prefWidth="572.0" AnchorPane.leftAnchor="14.0"
    AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="31.0">
      <children>
        <Label id="label1" text="MODE SELECTION:" AnchorPane.leftAnchor="14.0" AnchorPane.topAnchor="14.0" />
        <GridPane id="gridPane2" layoutX="142.0" prefHeight="139.0" prefWidth="191.0"
        AnchorPane.bottomAnchor="14.0" AnchorPane.topAnchor="16.0">
          <children>
            <RadioButton id="radioButton1" contentDisplay="CENTER" prefWidth="191.0" selected="true" text="OFF"
            GridPane.columnIndex="0" GridPane.rowIndex="0" />
            <RadioButton id="radioButton2" prefWidth="191.0" text="STDBY" GridPane.columnIndex="0"
            GridPane.rowIndex="1" />
            <RadioButton id="radioButton3" prefWidth="191.0" text="TST" GridPane.columnIndex="0"
            GridPane.rowIndex="2" />
            <RadioButton id="radioButton4" prefWidth="191.0" text="WXON" GridPane.columnIndex="0"
            GridPane.rowIndex="3" />
            <RadioButton id="radioButton5" prefWidth="190.9998779296875" text="WXA" GridPane.columnIndex="0"
            GridPane.rowIndex="4" />
          </children>
          <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
          </columnConstraints>
          <rowConstraints>
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          </rowConstraints>
        </GridPane>
        <Label id="label2" alignment="CENTER" prefWidth="54.0" text="WXR" AnchorPane.rightAnchor="2.0"
        AnchorPane.topAnchor="11.0" />
      </children>
    </AnchorPane>
    <AnchorPane id=" tiltSelection" layoutY="204.0" prefHeight="169.0" prefWidth="572.0"
    AnchorPane.leftAnchor="14.0" AnchorPane.rightAnchor="14.0">
      <children>
        <AnchorPane id="anchorPane3" prefHeight="80.0" prefWidth="197.0" AnchorPane.bottomAnchor="14.0"
        AnchorPane.rightAnchor="0.0">
          <children>
            <VBox id="TILTEdition" alignment="CENTER" prefHeight="80.0" prefWidth="89.0" spacing="10.0"
            AnchorPane.bottomAnchor="0.0" AnchorPane.rightAnchor="22.0" AnchorPane.topAnchor="0.0">
              <children>
                <Label id="label7" text="TILT ANGLE" />
```

```
            <TextField id="textField1" alignment="CENTER" prefWidth="82.0" text="+0°" />
          </children>
        </VBox>
      </children>
    </AnchorPane>
    <FlowPane id="SelectionCTRL" hgap="10.0" prefHeight="62.0" prefWidth="572.0" AnchorPane.leftAnchor="0.0"
    AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
      <children>
        <Label id="label3" text="TILT SELECTION:" />
        <Button id="btnCTRLSelect" text="CTRL" />
        <StackPane id="stackPane1" prefHeight="31.0" prefWidth="54.0">
          <children>
            <Label fx:id="lblSelect" text="AUTO" />
            <Line id="line1" endX="100.81915283203125" endY="-0.6391582489013672" startX="61.011474609375"
            startY="-0.9413433074951172" />
          </children>
        </StackPane>
      </children>
      <padding>
        <Insets left="10.0" top="10.0" />
      </padding>
    </FlowPane>
    <VBox id="StabCTRL" alignment="CENTER" prefHeight="121.0" prefWidth="100.0" spacing="10.0"
    AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0">
      <children>
        <Label id="label5" text="STABILIZATION" />
        <Button id="btnCTRLStab" disable="true" text="CTRL" />
        <Label id="lblStab" text="OFF" />
      </children>
    </VBox>
  </children>
</AnchorPane>
```

Annexe B. FXML code of the angle selector component

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>
<?import javafx.scene.shape.*?>

<AnchorPane id="AnchorPane" maxHeight="80.0" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="80.0" prefWidth="183.0" xmlns:fx="http://javafx.com/fxml" fx:controller="fr.irit.aldabra.mpia.
ICOAngleController">
  <children>
   <Line id="baseLine" endX="125.0" endY="-1.0" layoutX="26.0" layoutY="40.0" startX="-19.0" strokeWidth="3.0" />
    <Line fx:id="arrow" endX="50.39484786987305" endY="1.0253764390945435" layoutX="96.0" layoutY="30.0"
    rotate="355.0" startX="-88.0" startY="4.0" stroke="DODGERBLUE" strokeWidth="3.0" />
    <Arc id="arc1" fill="#1e90ff53" layoutX="24.0" layoutY="40.0" length="60.0" radiusX="100.0" radiusY="100.0"
   scaleX="1.3412279806726912" scaleY="0.5369465863735433" startAngle="330.0" stroke="BLACK" strokeType="INSIDE"
    strokeWidth="0.0" type="ROUND" />

    <Label fx:id="angleValue" layoutX="149.0" layoutY="15.0" text="+8°" />
    <Label id="label1" layoutX="107.0" text="+15°" />
    <Label id="label2" layoutX="107.0" layoutY="64.0" text="-15°" />
  </children>
  <stylesheets>
    <URL value="@defaultAngleStyleSheet.css" />
  </stylesheets>
</AnchorPane>
```

Annexe C. FXML code of MIPA including the angle selector

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>
<?import javafx.scene.shape.*?>

<AnchorPane id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="400.0" prefWidth="316.0" xmlns:fx="http://javafx.com/fxml">
  <children>
    <GridPane id="gridPane1" prefWidth="599.9999000000025" AnchorPane.bottomAnchor="0.0"
    AnchorPane.leftAnchor="1.0" AnchorPane.rightAnchor="1.0">
      <children>
        <ToggleButton id="toggleButton1" prefWidth="200.0" selected="true" text="WXR" GridPane.columnIndex="0"
        GridPane.rowIndex="0" />
        <ToggleButton id="toggleButton2" prefWidth="200.0" text="GCAS" GridPane.columnIndex="1"
        GridPane.rowIndex="0" />
        <ToggleButton id="toggleButton3" prefWidth="200.0" text="AIRCOND" GridPane.columnIndex="2"
        GridPane.rowIndex="0" />
      </children>
      <columnConstraints>
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
        <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
      </columnConstraints>
      <rowConstraints>
        <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      </rowConstraints>
    </GridPane>
    <AnchorPane id="header" prefHeight="31.0" prefWidth="598.0" AnchorPane.leftAnchor="1.0"
    AnchorPane.rightAnchor="1.0" AnchorPane.topAnchor="0.0">
      <children>
       <Label id="lblAUX" alignment="CENTER" prefHeight="29.0" prefWidth="89.0" text="AUX" textAlignment="CENTER"
        AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.topAnchor="0.0" />
      </children>
    </AnchorPane>
    <Separator id="separator" layoutY="201.0" prefWidth="569.0" AnchorPane.leftAnchor="15.0"
    AnchorPane.rightAnchor="16.0" />
    <AnchorPane id="anchorPane1" prefHeight="169.0" prefWidth="572.0" AnchorPane.leftAnchor="14.0"
    AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="31.0">
      <children>
        <Label id="label1" text="MODE SELECTION:" AnchorPane.leftAnchor="14.0" AnchorPane.topAnchor="14.0" />
        <GridPane id="gridPane2" layoutX="142.0" prefHeight="139.0" prefWidth="191.0"
        AnchorPane.bottomAnchor="14.0" AnchorPane.topAnchor="16.0">
          <children>
            <RadioButton id="radioButton1" contentDisplay="CENTER" prefWidth="191.0" selected="true" text="OFF"
            GridPane.columnIndex="0" GridPane.rowIndex="0" />
            <RadioButton id="radioButton2" prefWidth="191.0" text="STDBY" GridPane.columnIndex="0"
            GridPane.rowIndex="1" />
            <RadioButton id="radioButton3" prefWidth="191.0" text="TST" GridPane.columnIndex="0"
            GridPane.rowIndex="2" />
            <RadioButton id="radioButton4" prefWidth="191.0" text="WXON" GridPane.columnIndex="0"
            GridPane.rowIndex="3" />
            <RadioButton id="radioButton5" prefWidth="190.9998779296875" text="WXA" GridPane.columnIndex="0"
            GridPane.rowIndex="4" />
          </children>
          <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="100.0" />
          </columnConstraints>
          <rowConstraints>
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          </rowConstraints>
        </GridPane>
        <Label id="label2" alignment="CENTER" prefWidth="54.0" text="WXR" AnchorPane.rightAnchor="2.0"
        AnchorPane.topAnchor="11.0" />
      </children>
    </AnchorPane>
```

```
<AnchorPane id="anchorPane2" layoutY="204.0" prefHeight="169.0" prefWidth="572.0" AnchorPane.leftAnchor="14.0"
 AnchorPane.rightAnchor="14.0">
   <children>
     <AnchorPane id="anchorPane3" minWidth="182.0" prefHeight="80.0" prefWidth="182.0"
     AnchorPane.bottomAnchor="14.0" AnchorPane.rightAnchor="0.0">
       <children>
         <fx:include fx:id="TILTEdfition" source="TiltAngleSelector.fxml" AnchorPane.bottomAnchor="0.0"
         AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
       </children>
     </AnchorPane>
     <FlowPane id="SelectionCTRL" hgap="10.0" prefHeight="62.0" prefWidth="572.0" AnchorPane.leftAnchor="0.0"
     AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
       <children>
         <Label id="label3" text="TILT SELECTION:" />
         <Button id="btnCTRLSelect" text="CTRL" />
         <StackPane id="stackPane1" prefHeight="31.0" prefWidth="54.0">
           <children>
             <Label fx:id="lblSelect" text="AUTO" />
             <Line id="line1" endX="100.81915283203125" endY="-0.6391582489013672" startX="61.011474609375"
             startY="-0.9413433074951172" />
           </children>
         </StackPane>
       </children>
       <padding>
         <Insets left="10.0" top="10.0" />
       </padding>
     </FlowPane>
     <VBox id="StabCTRL" alignment="CENTER" prefHeight="121.0" prefWidth="100.0" spacing="10.0"
     AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0">
       <children>
         <Label id="label5" text="STABILIZATION" />
         <Button id="btnCTRLStab" disable="true" text="CTRL" />
         <Label id="lblStab" text="OFF" />
       </children>
     </VBox>
   </children>
 </AnchorPane>
</children>
<stylesheets>
 <URL value="@defaultMPIAStyleSheet.css" />
</stylesheets>
</AnchorPane>
```

References

[1] Android http://developer.android.com/.
[2] E. Anson, The device model of interaction, in: Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '82, Boston, Massachusetts, United States, July 26–30, 1982, ACM, New York, NY, 1982, pp. 107–114.
[3] C. Appert, M. BeaudouinLafon, SwingStates: adding state machines to the swing toolkit, in: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology, UIST '06, Montreux, Switzerland, October 15–18, 2006, ACM, New York, NY, 2006, pp. 319–322.
[4] ARINC 661, Prepared by Airlines Electronic Engineering Committee, Cockpit Display System Interfaces to User Systems, ARINC Specification 661, 2002.
[5] ARINC 661-2, Prepared by Airlines Electronic Engineering Committee, Cockpit Display System Interfaces to User Systems, ARINC Specification 661-2, 2005.
[6] E. Barboni, S. Conversy, D. Navarre, P. Palanque, Model-based engineering of widgets, user applications and servers compliant with ARINC 661 specification, in: Proceedings of the 13th Conference on Design Specification and Verification of Interactive Systems, DSVIS, in: LNCS, Springer Verlag, 2006.
[7] E. Barboni, J.-F. Ladry, D. Navarre, P. Palanque, M. Winckler, Beyond modelling: an integrated environment supporting co-execution of tasks and systems models, in: Proc. of the ACM SIGCHI conference Engineering Interactive Computing Systems, EICS 2010, Berlin, Germany, June 19–23, ACM SIGCHI, 2010, pp. 143–152.
[8] E. Barboni, D. Navarre, P. Palanque, S. Basnyat, A formal description technique for interactive cockpit applications compliant with ARINC specification 661, in: Proceedings of SIES 2007 — IEEE 2th International Symposium on Industrial Embedded Systems July 4–6, 2007, Lisbon, Portugal.
[9] L. Bass, et al., A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop, SIGCHI Bulletin 24 (1) (1992) 32–37.
[10] R. Bastide, O. Sy, P. Palanque, A formal notation and tool for the engineering of CORBA systems, in: Concurrency: Practice and Experience (Wiley). Special issue "Selected papers from ECOOP'99" vol. 12, 2000, pp 1379–1403.
[11] R. Bastide, P. Palanque, D.H. Le, J. Muñoz, Integrating rendering specifications into a formalism for the design of interactive systems, in: Proceedings of Design Specification and Verification of Interactive Systems' 98, DSVIS'98, Springer, Verlag, 1998, pp. 171–191.
[12] R. Bastide, P. Palanque, Petri nets with objects for specification, design and validation of user-driven interfaces, in: Proceedings of the Third IFIP TC13 Conference on Human-Computer Interaction, Interact'90.
[13] R. Blanch, M. Beaudouin-Lafon, Programming rich interactions using the hierarchical state machine toolkit, in: Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '06, Venezia, Italy, May 23–26, 2006, ACM, New York, NY, 2006, pp. 51–58.
[14] M. Book, V. Gruhn, Fine-grained specification and control of data flows in web-based user interfaces, Journal of Web Engineering (JWE) 8 (1) (2009) 48–70. Rinton Press, Paramus, NJ, USA.
[15] W. Buxton, A three-state model of graphical input, in: D. Diaper, D.J. Gilmore, G. Cockton, B. Shackel (Eds.), Proceedings of the IFIP Tc13 Third Interational Conference on Human-Computer interaction, August 27–31, 1990, North-Holland Publishing Co., Amsterdam, The Netherlands, 1990, pp. 449–456.

[16] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J.A. Vanderdonckt, Unifying reference framework for multi-target user interfaces, Interacting With Computers 15/3 (2003) 289–308.
[17] L. Cardelli, R. Pike, Squeak: a language for communicating with mice, SIGGRAPH Computer Graphics 19 (3) (1985) 199–204.
[18] D.A. Carr, Specification of interface interaction objects, in: B. Adelson, S. Dumais, J. Olson (Eds.), Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, CHI '94, Boston, USA, April 24–28, 1994, ACM, New York, NY, 1994, pp. 372–378.
[19] K. Coninx, E. Cuppens, J. De Boeck, C. Raymaekers, Integrating support for usability evaluation into high level interaction descriptions with NiMMiT, in: Interactive Systems: Design, Specification, and Verification. 2007, in: LNCS, Springr, Verlag, 2007, pp. 95–108.
[20] S. Conversy, E. Barboni, D. Navarre, P. Palanque, Improving modularity of interactive software with the MDPC architecture, in: Proceedings of EIS (Engineering Interactive Systems) Conference 2007, Joint HCSE 2007, EHCI 2007 and DSVIS 2007 Conferences, in: Lecture Notes in Computer Science, Springer Verlag, 2008.
[21] A.J. Dix, Formal Methods for Interactive Systems, s.l: Academic Press, 1991, 0-12-218315-0.
[22] P. Dragicevic, J. Fekete, Support for input adaptability in the ICON toolkit, in: Proceedings of the 6th International Conference on Multimodal Interfaces, ICMI '04, State College, PA, USA, October 13–15, 2004, ACM, New York, NY, 2004, pp. 212–219.
[23] B. Dumas, D. Lalanne, D. Guinard, R. Koenig, R. Ingold, Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces, in: Proceedings of the 2nd International Conference on Tangible and Embedded Interaction, TEI '08, Bonn, Germany, February 18–20, 2008, ACM, New York, NY, 2008, pp. 47–54.
[24] O. Esteban, S. Chatty, P. Palanque, Whizz'Ed: a visual environment for building highly interactive interfaces, in: Proceedings of the Interact'95 Conference, 1995, pp. 121–126.
[25] Flex, http://www.adobe.com/fr/products/flex.html.
[26] H.J. Genrich, Predicate/transitions nets, in: K. Jensen, G. Rozenberg (Eds.), High-Levels Petri Nets: Theory and Application, Springer, Verlag, Berlin, 1991, pp. 3–43.
[27] J. Guerrero-Garcia, J.M. Gonzalez-Calleros, J. Vanderdonckt, J. Munoz-Arteaga, A theoretical survey of user interface description languages: preliminary results, in: Proceedings of the 2009 Latin American Web Congress, LA-WEB '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 36–43. http://dx.doi.org/10.1109/LA-WEB.2009.40.
[28] J. Guerrero-García, J. Vanderdonckt, J.M. González-Calleros, FlowiXML: a step towards designing workflow management systems, International Journal of Web Engineering and Technology 4 (2) (2008) 163–182.
[29] K. Hinckley, M. Czerwinski, M. Sinclair, Interaction and modeling techniques for desktop two-handed input, in: Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, UIST '98, San Francisco, USA, November 01–04, 1998, ACM, New York, NY, 1998, pp. 49–58.
[30] S.E. Hudson, Graphical specification of flexible user interface displays, in: Proceedings of the 2nd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, UIST '89, Williamsburg, Virginia, United States, November 13–15, 1989, ACM, New York, NY, 1989, pp. 105–114.
[31] S.E. Hudson, J. Mankoff, I. Smith, Extensible input handling in the subArctic toolkit, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '05, Portland, Oregon, USA, April 02–07, 2005, ACM, New York, NY, 2005, pp. 381–390.
[32] HTML 5.0, http://www.w3.org/TR/html5/.
[33] R.J Jacob, A specification language for direct-manipulation user interfaces, ACM Transactions on Graphics 5 (4) (1986) 283–317.
[34] Java FX, http://javafx.com.
[35] K. Katsurada, Y. Nakamura, H. Yamada, T. Nitta, XISL: a language for describing multimodal interaction scenarios, in: Proceedings of the 5th International Conference on Multimodal Interfaces, ICMI '03, Vancouver, Canada, November 05–07, 2003, ACM, New York, NY, 2003, pp. 281–284.
[36] H.C. Keh, T.G. Lewis, Direct-manipulation user interface modeling with high-level Petri nets, in: Proceedings of the 19th Annual Conference on Computer Science, CSC '91, San Antonio, Texas, United States, ACM, New York, NY, 1991, pp. 487–495.
[37] D. Kieras, P.G. Polson, A generalized transition network representation for interactive systems, in: A. Janda (Ed.), Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '83, Boston, USA, December 12–15, 1983, ACM, New York, NY, 1983, pp. 103–106.
[38] J-F Ladry, D Navarre, P. Palanque, Formal description techniques to support the design, construction and evaluation of fusion engines for SURE (Safe Usable, Reliable and Evolvable) multimodal interfaces, in: International Conference on Multimodal Interfaces and Workshop on Machine Learning for Multi-modal Interaction, ICMI-MLMI 2009, ACM, Cambridge, Massachusetts, USA, 2009, pp. 135–142. 02/11/2009-06/11/2009.
[39] M.E. Latoschik, Designing transition networks for multimodal VR-interactions using a markup language, Multimodal Interfaces, 2002, in: Proceedings of the Fourth IEEE International Conference, 2002, pp. 411–416.
[40] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, López Jaquero Víctor, UsiXML: a language supporting multi-path development of user interfaces, in: Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004, Hamburg, July 11–13, in: LNCS, vol. 3425, Springer, 2004, pp. 200–220.
[41] C. Martinie, P. Palanque, D. Navarre, E. Poupart, A systematic approach to training for ground segment using tasks and scenarios: application to PICARD satellite, in: 12th International Conference on Space Operations, Space Ops 2012, June 11–15th, Stockholm, Sweden.
[42] C. Martinie, P. Palanque, D. Navarre, E. Barboni, A development process for usable large scale interactive critical systems: application to satellite ground segments, in: 4th International Conference on Human-Centred Software Engineering, HCSE, in: LNCS, Springer, Verlag, 2012.
[43] F. Montero, V. López-Jaquero, Gonzalez Vanderdonckt, P. Lozano, Solving the mapping problem in user interface design by seamless integration in IdealXML, in: S.W. Gilroy, M.D. Harrison (Eds.), Proc. of DSV-IS'2005, Newcastle upon Tyne, 13–15 July 2005, in: Lecture Notes in Computer Science, vol. 3941, Springer-Verlag, Berlin, 2005, pp. 161–172.
[44] B. Michotte, J. Vanderdonckt, GrafiXML, a multi-target user interface builder based on UsiXML, in: Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS'2008, Gosier, 16–21 March 2008, IEEE Computer Society Press, Los Alamitos, 2008, pp. 15–22.
[45] D. Navarre, P. Palanque, R. Bastide, A formal description technique for the behavioural description of interactive applications compliant with ARINC 661 specifications, HCI-Aero'04 Toulouse, France, 29 September–1st October 2004.
[46] D. Navarre, P. Palanque, R. Bastide, O. Sy, A model-based tool for interactive prototyping of highly interactive applications, in: 12th IEEE, International Workshop on Rapid System Prototyping; Monterey (USA), IEEE, 2001.
[47] D. Navarre, P. Palanque, J.F. Ladry, E. Barboni, ICOs: a model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability, Journal ACM Transactions on Computer-Human Interaction (TOCHI) 16 (4) (2009). ACM New York, NY, USA.
[48] D. Navarre, P. Palanque, J.F. Ladry, S. Basnyat, An architecture and a formal description technique for user interaction reconfiguration of safety critical interactive systems, in: The XVth International Workshop on the Design, Verification and Specification of Interactive Systems, DSVIS 2008, Kingston, Ontario, Canada, July 16–18 2008.
[49] D. Navarre, P. Palanque, R. Bastide, A. Schyn, M. Winckler, L. Nedel, C. Freitas, A formal description of multimodal interaction techniques for immersive virtual reality applications, in: Proceedings of INTERACT 2005, Roma, Italy, September, in: Lecture Notes in Computer Science, Springer Verlag, 2005.
[50] Openlaszlo, http://www.openlaszlo.org/.
[51] R. Schaefer, S. Bleul, W. Mueller, Dialog modeling for multiple devices and multiple interaction modalities, in: Karin Coninx, Kris Luyten, Kevin A. Schneider (Eds.), Proceedings of the 5th International Conference on Task Models and Diagrams for Users Interface Design, TAMODIA'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 39–53.
[52] O. Shaer, M. Green, R.J.K Jacob, K. Luyten, User interface description languages for next generation user interfaces, in: Proc. of Extended Abstracts of CHI'08, ACM Press, New York, 2008, pp. 3949–3952.
[53] J.L. Sibert, W.D. Hurley, T.W Bleser, An object-oriented user interface management system, in: D.C. Evans, R.J. Athay (Eds.), Proceedings of the 13th Annual Conference on Computer Graphics and interactive Techniques, SIGGRAPH '86, ACM, New York, NY, 1986, pp. 259–268.
[54] C.E. Silva, J.C. Campos, Can GUI implementation markup languages be used for modelling, in: Human Centred Software Engineering, HCSE 2012, Toulouse, France, November 2012.
[55] Silverlight, http://www.microsoft.com/silverlight/.

[56] S. Smith, D. Duke, Using CSP to Specify Interaction in Virtual Environments, Technical Report YCS 321, University of York, 1999.
[57] SVG W3C 2003: Scalable Vector Graphics (SVG) 1.1 Specification http://www.w3.org/TR/SVG11/.
[58] P. Szekely, B. Myers, A user interface toolkit based on graphical objects and constraints, in: N. Meyrowitz (Ed.), Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, OOPSLA '88, ACM, New York, NY, 1988, pp. 36–45.
[59] K. Tatsukawa, Graphical toolkit approach to user interaction description, in: S.P. Robertson, G.M. Olson, J.S. Olson (Eds.), Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '91, ACM, New York, NY, 1991, pp. 323–328.
[60] V. Tran, J. Vanderdonckt, M. Kolp, Y. Wautelet, Using task and data models for user interface declarative generation, in: J. Filipe, J. Cordeiro (Eds.), Proc. of 12th International Conference on Enterprise Information Systems ICEIS'2010, Funchal, 8–10 June 2010, vol. 5, SciTePress, 2010, pp. 155–160.
[61] F.M. Trindade, M.S. Pimenta, RenderXML — A Multi-platform Software Development Tool, TAMODIA, in: LNCS, vol. 4849, Springer, 2007, pp. 293–298.
[62] J. Vanderdonckt, Q. Limbourg, M. Florins, Deriving the navigational structure of a user interface, in: Proc. of Interact' 2003, Zurich, 1–5 September 2003, IOS Press, Amsterdam, 2003, pp. 455–462.
[63] J.S. Willans, M.D Harrison, Prototyping pre-implementation designs of virtual environment behaviour, in: M.R. Little, L. Nigay (Eds.), Proceedings of the 8th IFIP International Conference on Engineering For Human-Computer Interaction, May 11–13, 2001, in: Lecture Notes In Computer Science, vol. 2254, Springer-Verlag, London, 2001, pp. 91–108.
[64] M. Winckler, F.M. Trindade, A. Stanciulescu, J. Vanderdonckt, Cascading dialog modeling with UsiXML, in: Proc. of 15th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2008, Kingston, July 16–18, 2008, in: Lecture Notes in Computer Sciences, vol. 5136, Springer, Berlin, 2008, pp. 121–135.
[65] M. Winckler, P. Palanque, StateWebCharts: a formal description technique dedicated to navigation modelling of web applications, in: International Workshop on Design, Specification and Verification of Interactive Systems — DSVIS'2003, Funchal, Portugal, June 2003.
[66] World Wide Web Consortium, State Chart XML (SCXML): State Machine Notation for Control Abstraction, Working Draft 26 April 2011. Available at: http://www.w3.org/TR/2011/WD-scxml-20110426/.
[67] XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999. Available at: http://www.w3.org/TR/xslt.
[68] XUL (XML User Interface Language). Available at: http://www.mozilla.org/projects/xul/ (August 10, 2011).